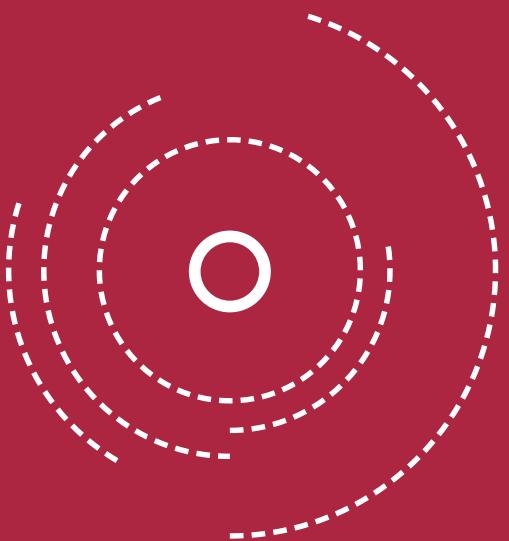


Mojaloop Comprehensive Documentation

2018-10-30



This work is copyright © 2017 Bill & Melinda Gates Foundation and made available under a [Creative Commons Attribution-ShareAlike 4.0 International License](#)



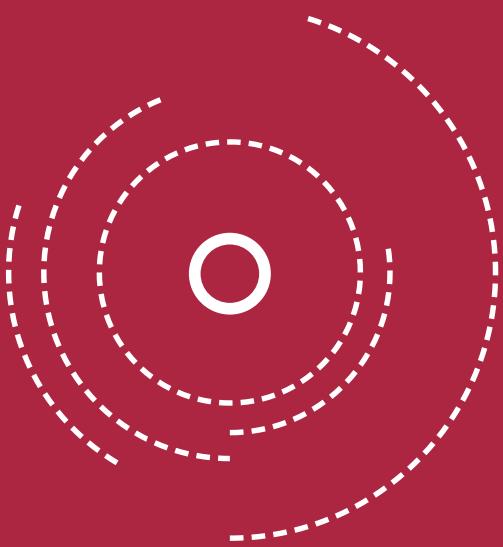


Table of Contents

Overview

- [Mojaloop Overview](#)
- [Documentation Overview](#)
- [Central Services Overview](#)

Scenarios

- [Scenario Descriptions](#)
- [Pending Transfers](#)
- [Pending Transfers API](#)
- [Bulk Payments](#)

Level One Client

- [Interledger Components and Flow](#)
- [ILP Service](#)
- [Interop Services](#)
- [Interop Ledger](#)

Central Directory

- [Central Directory](#)
- [User Discovery](#)
- [User Retrieval Guide](#)
- [Central Directory API](#)
- [Directory Gateway](#)

Central Ledger

- [Central Ledger](#)
- [Central Ledger API](#)
- [Working with transfers](#)
- [Central ledger Configuration](#)
- [Central Ledger Setup](#)
- [Running Central Ledger on Kubernetes](#)

Central Rules

- [Central Rules](#)
- [Central Rules Endpoints](#)

Central Fraud

- [Central Fraud Sharing API Documentation](#)

Infrastructure

- [Setup Logging](#)
- [Using Kibana](#)
- [Using Logging](#)
- [Forensic Logging Sidecar Guide](#)
- [Mule's Docker Image](#)
- [Configuring High Availability Proxy](#)

Testing

- [Use Case Tests](#)
- [Account Management Tests](#)
- [Customer Management Tests](#)

- DFSP Management Tests
- Fees Tests
- Pending Transaction Tests
- Send Money Tests
- Scenario Automation
- Integration Automation
- Resilience Modeling and Anaylysis

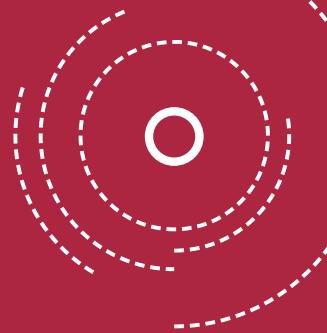
Security

- PKI Guide

Appendix

- Terminology
- Tools and Process Decisions
- Exporting the Documents

Overview



Getting Started

Mojaloop aims at creating payment platforms that are interoperable, connecting digital financial service providers and customers by providing specifications, standards and open-source software. The Mojaloop project is organized on the basis of component microservices. As such, there are over twenty different repositories in GitHub that align to the different services. The following four repositories are pinned to the project: - The [mojaloop repository](#) contains this master readme file along with the following documents to get started. * [contributors guide](#) for development and installation materials. * [frequently asked questions](#). - The [docs repository](#) documents the overall architecture, component design, message flow, and an overview of Mojaloop. - The [project repository](#) is the central repository to track product development issues for the Mojaloop project. - The [mojaloop-specification](#) contains the specification document set of the Open API for Financial Service Provider Interoperability.

- Individual repositories in the [Mojaloop GitHub organization](#) each describe component-specific details including source and APIs.

Mojaloop Overview

[chat on gitter](#) Digital and mobile technologies make it possible to reach new customers in developing markets with innovative, low-cost financial services. The lack of a shared platform, though, means that financial providers have to build everything on their own. This raises costs and keeps the digital financial industry from growing and innovating as fast as it could.

Mojaloop is open-source software for creating digital payments platforms that connect all customers, merchants, banks, and other financial providers in a country's economy. Rather than a financial product or application in itself, Mojaloop establishes a blueprint for technology that bridges all the financial products and applications in any given market.

The intention is for financial institutions and commercial providers to use the open-source software to help build digital, interoperable payments platforms that drive financial inclusion on a national scale. Specifically, the platforms will enable seamless, low-cost transactions between individual users, merchants, banks, providers, and even

government offices - helping connect poor customers with everyone else in the digital economy.

Mojaloop grew out of principles set forth by the Financial Services for the Poor team at the Bill & Melinda Gates Foundation. These principles form the cornerstone of the Level One Project, which is the foundation's initiative for designing and implementing digital financial services and systems to include and benefit the world's 2 billion unbanked. The code won't address the gap in digital financial services on its own, but it can help service providers open markets and accelerate progress.

With support and funding from the Bill & Melinda Gates Foundation, Mojaloop was designed by a team of leading tech and fintech companies: [Ripple](#), [Dwolla](#), [ModusBox](#), [Software Group](#) and [Crosslake Technologies](#). It is available now as an open-source project. Free to be used and adapted by anyone under the [Apache 2.0 license](#). For more information on how Mojaloop ties into the larger initiative of financial inclusion, visit [LevelOneProject.org](#).

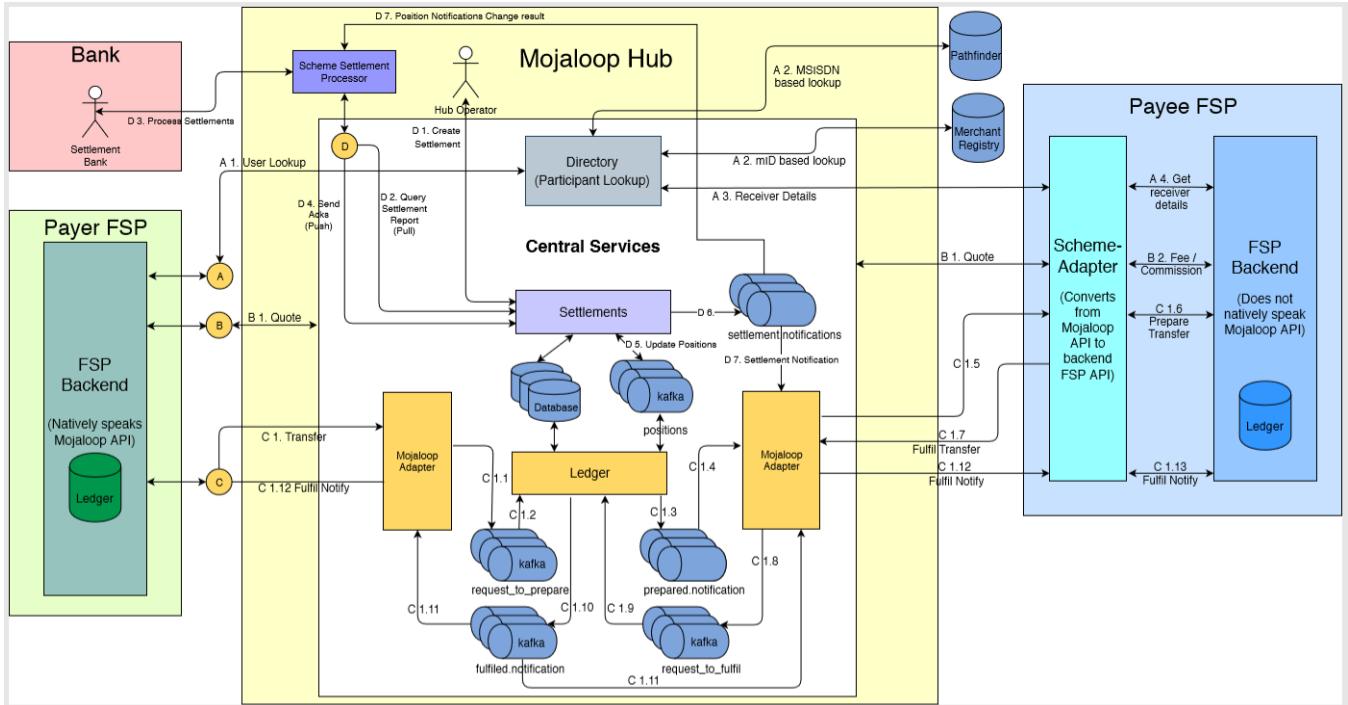
Why should I contribute to Mojaloop?

Mojaloop makes it easier for financial providers and entrepreneurs to go to market with products and services, especially in developing markets where demand is high and supply is low.

A nationwide interoperable digital payments platform requires really good code. Your contributions will help ensure that Mojaloop covers all the bases, translates well to real-world scenarios, and meets the many and diverse needs of financial providers and customers.

Mojaloop Services

The following architecture diagram shows the Mojaloop services:



The basic idea behind Mojaloop is that we need to connect multiple Digital Financial Services Providers (DFSPs) together into a competitive and interoperable network in order to maximize opportunities for poor people to get access to financial services with low or no fees. We don't want a single monopoly power in control of all payments in a country, or a system that shuts out new players. It also doesn't help if there are too many isolated subnetworks. Our model addresses these issues in several key ways:

- A set of central services provides a hub through which money can flow from one DFSP to another. This is similar to how money moves through a central bank or clearing house in developed countries. Besides a central ledger, central services can provide identity lookup, fraud management, and enforce scheme rules.
- A standard set of interfaces a DFSP can implement to connect to the system, and example code that shows how to use the system. A DFSP that wants to connect up can adapt our example code or implement the standard interfaces into their own software. The goal is for it to be as straightforward as possible for a DFSP to connect to the interoperable network.
- Complete working open-source implementations of both sides of the interfaces - an example DFSP that can send and receive payments and the client that an existing DFSP could host to connect to the network.

What's here and what's not

This is free code provided under an [Apache 2.0 license](#).

The code is released with an Apache 2.0 license but the Specification documents under the 'mojaloop-specification' documents are published with CC BY-ND 4.0 License

We don't provide production servers to run it on. That's up to you. You are free (and encouraged!) to clone these repositories, participate in the community of developers, and contribute back to the code.

We are not trying to replace any mobile wallet or financial providers. We provide code to link together new and existing financial providers using a common scheme. There are central services for identifying a customer's provider, quoting, fulfillment, deferred net settlement, and shared fraud management. Each provider can take advantage of these services to send and receive money with others on the system and there's no cost to them to onboard new providers. We provide code for a simple example mobile money provider to show how integration can be done, but our example DFSP is not meant to be a production mobile money provider.

Related Projects

The [Interledger Protocol Suite \(ILP\)](#) is an open and secure standard that enables DFSPs to settle payments with minimal *counter-party risk* (the risk you incur when someone else is holding your money). With ILP, you can transact across different systems with no chance that someone in the middle disappears with your money. Mojaloop uses the Interledger Protocol Suite for the clearing layer. For an overview of how it works, see the [Clearing Architecture Documentation](#).

Docs Overview

The *docs* repository documents the overall architecture, component design, message flow, high level tests and an overview of the Mojaloop software.

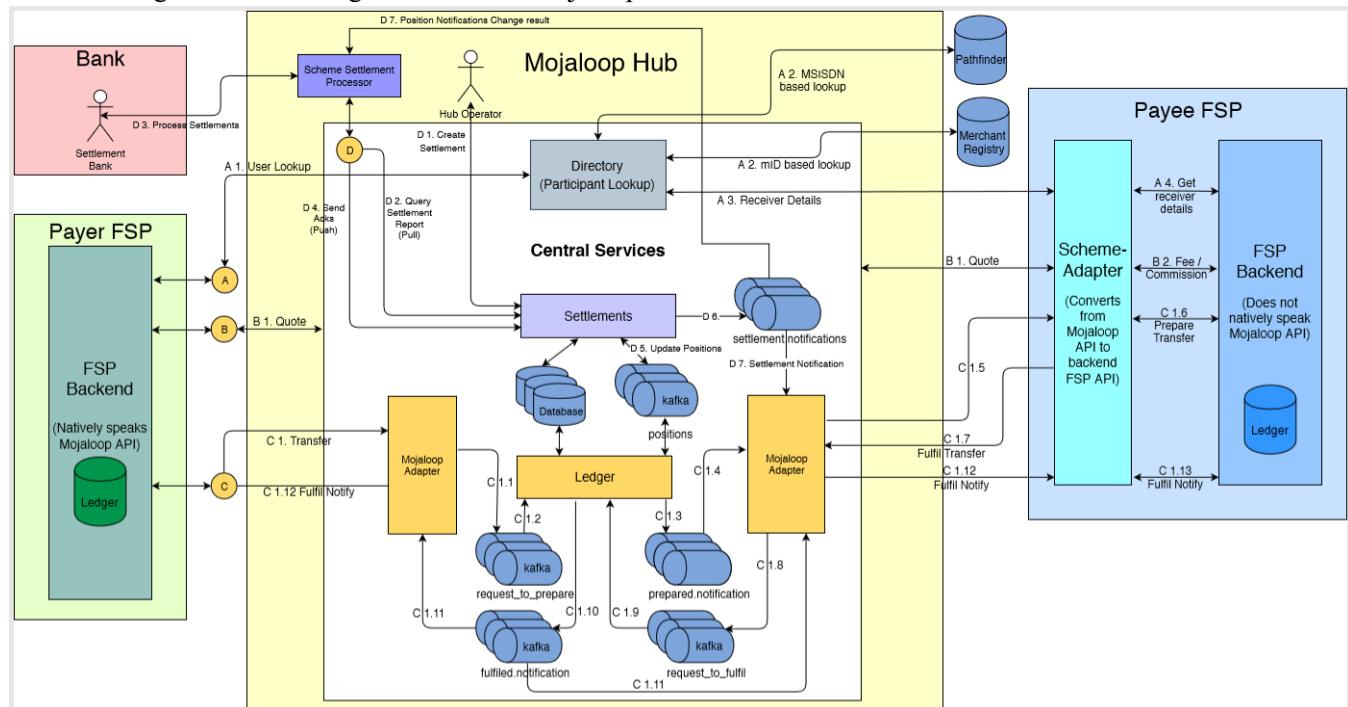
Individual repositories in the [mojaloop GitHub organization](#) each describe component-specific details including source and APIs.

For more information on mojaloop, see the <https://mojaloop.io>

New developers, see the [contributors guide](#) for onboarding materials. [chat](#) [on gitter](#)

Mojaloop Services

The following architecture diagram shows the Mojaloop services:



ML-Adapter

The Mojaloop Adapter is the translation layer to convert to/from Mojaloop API to an internal format that is used in Central Services Stack.

Central Services

The central ledger is a series of services that facilitate clearing and settlement of transfers between DFSPs, including the following functions:

- Brokering real-time messaging for funds clearing
- Maintaining net positions for a deferred net settlement
- Propagating scheme-level and off-transfer fees

End-to-End Scenarios

The aforementioned individual services can't alone describe how key scenarios work across the system. Therefore, for each of the [Mojaloop Scenarios](#), we provide a technical walk through.

1. Send Money to Anyone: [scenario](#)
2. Buy Goods [scenario](#)
3. Bulk Payment [scenario](#)

Related Projects

The [Interledger Protocol Suite](#) (ILP) is an open and secure standard that enables DFSPs to settle payments with

minimal *counter-party risk* (the risk you incur when someone else is holding your money). With ILP, you can transact across different systems with no chance that someone in the middle disappears with your money. Mojaloop uses the Interledger Protocol Suite for the clearing layer. For an overview of how it works, see the [Clearing Architecture Documentation](#).

About This Document

This document is a work in progress; not all sections are updated to the latest developments in the project. Sections that are known to be out of date are marked as follows:

Out Of Date Content

Any text in this area is considered "out of date." It may reflect earlier versions of the technology, outdated terminology use, or sections that are poorly phrased and edited.

DFSP Service

The DFSP code is an example implementation of a mobile money provider. Customers connect to it from their mobile feature phones using Unstructured Supplementary Service Data (USSD). USSD is a Global System for Mobile (GSM) communication technology that is used to send text between a mobile phone and an application program in the network, allowing users to create accounts, send money, and receive money.

[DFSP Documentation](#)

Central Services

The central services are a collection of separate services that help the DFSPs perform operations on the network.

- The [Central Directory Service](#) determines which DFSP handles a user's accounts.
- The [Central Ledger Service](#) handles clearing and settlement.
- The [Central Rules Service](#) sets policy across the system.
- The [Fraud service](#) aids DFPS in identifying suspicious behavior.

Level One Client Service

The client service connects a DFSP to other other DFSPs and the central services. It has a few simple interfaces to

connect to a DFSP for account holder lookup, payment setup, and ledger operations. The level one client can be hosted locally by the DFSP or in a remote data center such as Amazon.

System-wide Testing

Individual services have their own tests, but the [testing strategy](#) also includes the following system-wide tests:

- [Scenario testing](#)
- [End-to-end functional testing](#)
- [Performance testing](#)
- [Resilience Modeling and Analysis \(RMA\)](#)
- Threat Modeling

End-to-End Scenarios

The aforementioned individual services can't alone describe how key scenarios work across the system. Therefore, for each of the [Mojaloop Scenarios](#), we provide a technical walk through.

1. Send Money to Anyone: [scenario](#), [walkthrough](#)
2. Buy Goods [scenario](#), [message flow](#)
3. Bulk Payment [scenario](#), [message flow](#)

Central Services Overview

The central services stack provides shared functions that allow scheme participants and Digital Financial Service Providers (DFSPs) to execute several actions using a consistent communication channel. In addition, the functions of the central services promote overall health of the scheme, allowing DFSPs to participate with confidence and reliability.

The information in this section summarizes the various services that the central services stack offers:

Directory

The central directory is a set of services that allows DFSPs to register and retrieve scheme identifiers. The scheme identifier can be leveraged by DFSPs for end-user discovery. The services, APIs and endpoints enable:

- Registering a DFSP
- Adding an end user
- Retrieving an end user

To view the references and available endpoints, please see the [Central Directory repository](#).

Ledger

The central ledger is a set of services that facilitate clearing and settlement of transfers between DFSPs, including the following functions:

- Brokering real-time messaging for funds clearing
- Maintaining net positions for a deferred net settlement
- Propagating scheme-level and off-transfer fees

To view the references and available endpoints, please see the [Central Ledger repository](#).

Fraud Sharing - *in-progress*

The fraud sharing service offers participating DFSPs an avenue to share end user and transactional information to help promote overall health of the scheme via fraud prevention, focusing on:

- Sharing end user and transaction information
- Enabling DFSPs to prevent fraud, not the scheme

The service is current being delivered and will be available as an initial product offering.

Forensic Logging - *in-progress*

The forensic logging solution allows information required to ensure the confidentiality and integrity of the overall central services stack. Events are captured, preserved and made available to authorized inquiries. Functions of the logging mechanism include:

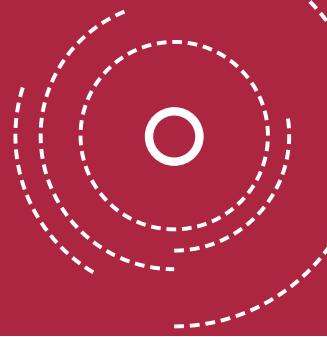
- Distributed implementation and log creation/storage
- Centralized Key Management Service (KMS)
- Cryptographic protection of data in-transit (encryption) including proof of integrity (signing)
- Removes a single point of failure

The service is current being delivered and will be available as an initial product offering.

Authentication

Currently the centralized services leverage basic authentication to secure interactions. A basic authentication solution was chosen to ensure a demonstration would be available while allowing adaptations for specific integrations. This solution is fully compatible with an HTTPS-based (TLS) environment.

Scenarios



About Mojaloop Scenarios

Mojaloop addresses a number of scenarios in which this software might help the poor to meet their financial and banking needs. There are several different paths for each of these scenarios, including potential timeout issues and reversals (which are handled as a separate transaction). The most common paths include:

- [Send money to anyone](#)
- [Buy Goods](#)
- [Bulk Payments](#)
- [Tiers/Risk Level](#)
- [Fraud Checks and Blacklists](#)
- [Account Management](#)
- [Check Account and POS](#)

Scenario Descriptions

Send money to anyone

Hamim is working on his farm in Northern Uganda when he receives an urgent phone call from his brother, Kani. Kani is low on money and will not get paid until next week. Kani has no money to pay for food for his family and asks Hamim to help him out. Hamim and Kani have no means of transportation and it would take several days for Hamim to get to his Kani's home in Southern Uganda. While they both have mobile flip phones, they use different financial service providers. Kani tells Hamim that he needs 5,000 shillings to buy food until he gets paid next week for his job working in a local field. Hamim agrees to send Kani the money.

*** Mojaloop technology does its job ***

Because Hamim has sent money to Kani before he has his information on his phone. Hamim sees Kani's name come up on his phone and he starts the transaction. He also sees the total fees and any exchange rates he has to pay before he sends the money. He is happy for that validation and that the transaction goes the same way every time. In under 30 seconds, Hamim is able to send the money to his Kani and verifies that he got it. Hamim is happy he was able to help out Kani and his family so quickly so they can buy food.

Buy Goods - Pending Transactions

Venya is waiting in line to buy plantains at her local market. She is corralling her elder child with one hand and has her baby in a sling. She often comes to this seller and she knows he has a good price. She also knows that even though she carries no money and he is not on her financial network, she can buy from him. As she approaches the head of the line she juggles the children and pulls out a simple flip phone. She tells him 1.5 kilograms and he tells her the price, which she agrees to.

*** Mojaloop technology does its job ***

Because she's been here before the merchant already has her information on his phone. The only information he has is her user number. This makes Venya feel safe that the merchant does not have her mobile phone number. The merchant enters in the amount for the plantains. Almost instantaneously, Venya sees the merchants invoice on her phone and she is glad she is able to pay for the transaction using her mWallet account. She is happy that the transaction goes the same way every time, because half of her attention is on the children. She has friends who can't read and they are able to buy things this way too by following the simple order of the transaction. In under 30 seconds, she is able to send the money to the merchant and both Venya and the merchant get confirmation of the transaction. She tells the elder child to pick up her plantains and makes room for the next person in line.

Bulk Payments

Nikisha is the accountant for one of the largest manufacturing companies in Johannesburg and employs over 250 workers. Their company uses a time and attendance system to track the number of hours that each employee works along with their hourly rate and employee ID. On a weekly basis Nikisha gets an update on her bulk payment system that shows all the employees, their user ID along and amount to be paid. Since the companies employees all have different financial service providers this system makes it really easy for Nikisha to confirm and distribute their pay with a couple of clicks. The company has a high turnover rate so new employees who get their first paycheck are automatically prompted to open an account as long as they provided a valid mobile number when they started. As Nikisha gets ready to send out this week's payments she opens up a bulk payment report.

*** Mojaloop technology does its job ***

The bulk report for payments comes up by date range and, since Nikisha does this weekly, there are several items she needs to verify each time. Specifically, she looks for any errors or alerts for employees with invalid phone numbers, names not matching or other anomalies. Nikisha has the ability to follow-up with her co-workers or employees directly to fix these errors before sending out the payments. In addition, Nikisha is also notified of any employees who don't have an account setup. For these users, Nikisha is still able to push a payment through and the employee will be

prompted by text message to open an account. Nikisha is thankful she has this process that makes it much easier to distribute funds. Once Nikisha has completed her validation, she sends it to her supervisor for final approval. Nikisha is glad to have this system in place because several years ago, Nikisha and her supervisor had to pay employees in cash and use a manual system to verify payments were received which made her feel very uneasy.

Tiers/Risk Levels

Salem works as an auditor for a large bank in Kampala, Uganda. His job is to monitor, manage and reduce risk for the company. In order to do so each new user ID in the system is assigned a default tier level which means they can only transfer a small number and amount of funds in and out of the system over specific periods of time. As users acquire greater balances in their accounts and hold their accounts for longer periods of time their tier levels improve. Tier levels are handled automatically by the system so Salem does not need to worry about assigning or changing these levels under normal circumstances. Part of Salem's job as an auditor is to review the daily reports to ensure that everyone's funds are safe and secure and he kicks off his daily report to review the accounts.

*** Mojaloop technology does its job ***

This morning when Salem reviews these reports he notices that one specific user ID has 32 outgoing transactions in one day which exceeds their daily count of 25. This seems very suspicious to Salem and he goes ahead and contacts the customer. It turns out that this customer is a local merchant that owns a store. The merchant explains that he has to go to the market on a weekly basis to get ingredients for his restaurant and it is not uncommon for him and his staff to make more than 25 purchases in one day. Although this customer has only been with Salem's bank for a month they have a healthy balance in their account. Salem goes ahead and upgrades the customers tier level to increase the daily and weekly transaction counts.

Fraud Checks and Blacklists

Salem works as an auditor for a large bank in Kampala, Uganda. His job is to monitor and stop any fraudulent activity for the company. While the company has a set of rules that might flag individuals for Salem to investigate, he also has the authority to screen any user ID for fraudulent activities at any time. Each time Salem performs a fraud check on a user ID, the system records the date of the last check along with any comments that Salem might have made. This makes it very easy for Salem to search for IDs that might have never been checked for fraud or have not been checked in a very long time. Salem has been monitoring one particular ID that seems to have had an increased amount of incoming funds deposited into their account on a daily basis. Today he does a search of the user ID to investigate further.

*** Mojaloop technology does its job ***

When the user ID is retrieved Salem is able to see the customer's name, birthdate and national ID number. He also sees any additional IDs and the account type associated with this customer. Upon further inspection Salem sees once again the number and amount of transactions deposited into this account has doubled again today. Salem suspects that this user is involved in some illegal activity and would like to send this up to his supervisor to get someone to do a deeper investigation. In the meantime to ensure that the illegal funds don't continue to come into the system, Salem decides to

'freeze' the account. Salem does this by checking the blacklist button and indicating a reason for the blacklist. At this point any future deposits or withdrawals for this User ID will be denied until someone from the Bank removes them from the blacklist. Salem feels good that no additional funds that might come from illegal or unapproved sources will be deposited into this customer account.

Account Management

Tadeo just bought his first mobile flip phone for him and his family to share. He is happy that he finally has a phone that he can use in emergencies, but he can also finally keep his money secure by opening up a bank account. Tadeo has never had a bank account since he lives in a very remote part of Africa with no personal means of transportation. Tadeo and his family have to rely on bartering or cash to buy any goods they need at the market. Although Tadeo is not proficient in reading, he is able to easily use his phone to setup and account for him and his family by following a couple of easy to read menu steps.

*** Mojaloop technology does its job ***

Tadeo was able to use his phone to create an mWallet account using his National ID. He was also asked to create a unique pin which made him feel secure in case him or someone in the family lost the phone. Tadeo is the primary account owner and he was able to easily create a new account for his oldest son. He was very pleased that he could have separate accounts for his son. His son is married and lives with Tadeo but does not have a phone. Since his son works it is only fair that they should be able to spend his money on goods and foods that he and his wife prefer. Tadeo also adds his wife as a user on his account. He allows his wife to be a signatory since she does most of the shopping at the local market and now has the ability to pay for goods using this phone. Tadeo is very happy that his wife no longer needs to have cash or carry barter goods to the market.

Check Account and POS

Jahari has a flip phone that all the family uses and he has setup different user numbers for each family member. Jahari is at the local market and needs to buy some meat for his family. Before he does, he wants to make sure he has enough funds in his account to purchase the goods and still have enough left over to set aside for future medical expenses and education. Jahari is happy that his money is secure and he is able to check his account balance anytime he needs to by simply entering his secure pin on his phone. Once he confirms his balance he will buy some goat and cow meat at the market.

*** Mojaloop technology does its job ***

After Jahari has entered his pin on his phone he is able to see his account balance. He is also able to see any of his recent transactions as well as any fees that were associated with these transactions. After confirming his available funds he picks out his meat and brings it up to the merchant for payment. The merchant does a lot of business in this market and has a point of sales device. This is very helpful for Jahari and his family since they only have one phone and many times his wife or his children go to the market and do not have the phone with them. The merchant is able to enter the purchase amount on the POS device and Jahari or any of his family members securely enters their user number and reviews the transaction. Jahari confirms that the amount on the POS machine matches what the merchant

verbally told him and he enters his pin to approve the transaction.

*** Mojaloop technology does its job ***

The merchant gets confirmation that he received payment and he prints a receipt for Jahari. Since Jahari has his phone with him today he also re-checks his account balance again to confirm that appropriate funds were taken from his account. Jahari is happy that this is an easy process and he can see that he has plenty of money left to set aside this month for his family to use on education or health expenses.

Pending Transaction

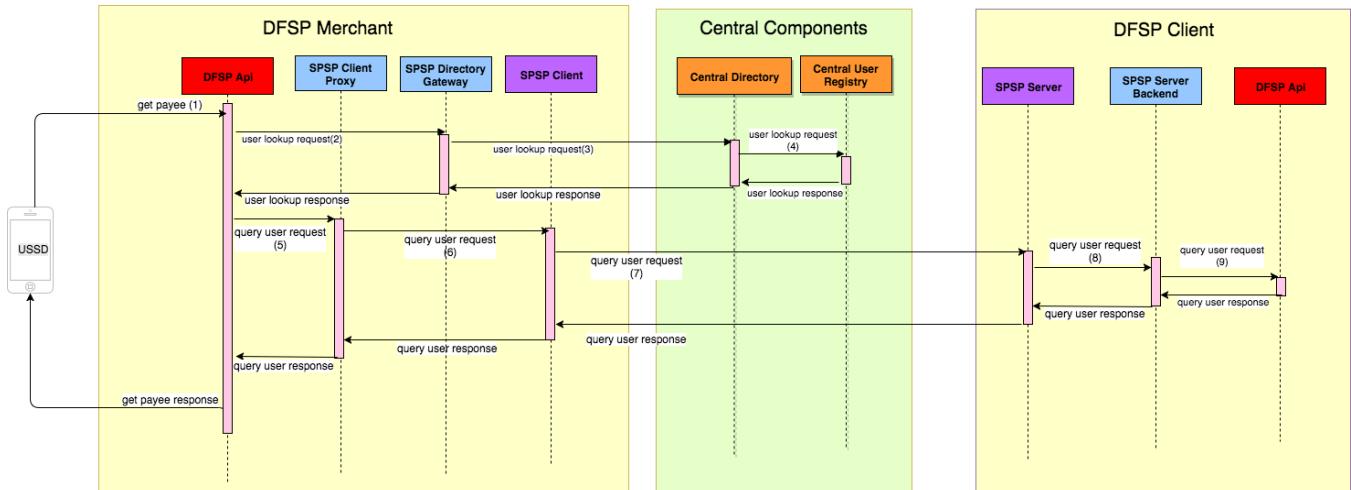
See [Scenario definition](#)

Assumptions

1. The invoice will be created in the merchant's DFSP. It will be associated with an account.
2. After the invoice is created in the merchant's DFSP, a notification with the invoice reference will be send to the primary client DFSP.
3. The client DFSP will stored the reference (full URL) to the merchant's invoice.
4. The invoice reference in the client DFSP will not be associated with any client's account; thus the client can choose the account from which he is going to pay the invoice.
5. As a consequence of the above, in case the client has accounts in more than one DFSP, he will receive the invoice notification only in his primary DFSP. From the USSD interface he will be able to pay the invoice only from his primary DFSP.

Note: 'dfsp1' is referred to as client DFSP (paying the invoice) and 'dfsp2' as the merchant DFSP (issuing the invoice).

I. Get Sender Details



DFSP USSD -> DFSP API

1.1 Get Payee

This method is not exposed as a DFSP Api rest route as it is not meant to be called directly from external systems.

Request:

```
{  
    "identifier": "78956562"  
}
```

Response: 200 OK

```
{  
    "spspReceiver": "http://dfsp2-spsp-server:3043/v1"  
}
```

- spspReceiver - the base url of sender's spsp server

DFSP API -> Directory Gateway

1.2 User Lookup Request

Request:

```
GET http://central-directory/  
resources?identifier=78956562&identifierType=eur
```

Response:

200 OK

```
{  
    "spspReceiver": "http://dfsp2-spsp-server:3043/v1",  
    "type": "payee",  
    "name": "bob",  
    "account": "levelone.dfsp2.bob",  
    "currencyCode": "USD",  
    "currencySymbol": "$",  
    "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/  
profile_pic.jpg"  
}
```

- spspReceiver - the base url of sender's spsp server

Directory Gateway -> Central Directory

1.3 User Lookup Request

Request:

```
GET http://central-directory/
resources?identifier=78956562&identifierType=eur
```

Response:

200 OK

```
{
  "spspReceiver": "http://dfsp2-spsp-server:3043/v1",
  "type": "payee",
  "name": "bob",
  "account": "levelone.dfsp2.bob",
  "currencyCode": "USD",
  "currencySymbol": "$",
  "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/
profile_pic.jpg"
}
```

- spspReceiver - the base url of sender's spsp server

Central Directory -> Central User Registry

1.4 User Lookup Request

-- to be filled in

DFSP API -> SPSP Client Proxy

1.5 Query User Request

Request:

```
GET http://dfsp1.spsp-client-proxy/spsp.client/v1/
query?receiver=http://dfsp2-spsp-server:3043/v1/receivers/78956562
```

Response:

200 OK

```
{  
    "type": "payee",  
    "name": "bob",  
    "account": "levelone.dfsp2.bob",  
    "currencyCode": "USD",  
    "currencySymbol": "$",  
    "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/  
profile_pic.jpg"  
}
```

SPSP Client Proxy -> SPSP Client

1.6 Query User Request

Request:

```
GET http://dfsp1.spdp-client-proxy/spdp.client/v1/  
query?receiver=http://dfsp2-spdp-server:3043/v1/receivers/78956562
```

Response:

200 OK

```
{  
    "type": "payee",  
    "name": "bob",  
    "account": "levelone.dfsp2.bob",  
    "currencyCode": "USD",  
    "currencySymbol": "$",  
    "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/  
profile_pic.jpg"  
}
```

SPSP Client -> SPSP Server

1.7 Query User Request

Request:

```
GET http://dfsp2-spdp-server:3043/v1/receivers/78956562
```

Response:

200 OK

```
{
```

```
"type": "payee",
"name": "bob",
"account": "levelone.dfsp2.bob",
"currencyCode": "USD",
"currencySymbol": "$",
"imageUrl": "https://red.ilpdemo.org/api/receivers/bob/
profile_pic.jpg"
}
```

SPSP Server -> SPSP Server Backend

1.8 Query User Request

-- to be filled in

SPSP Server Backend -> DFSP Api

1.9 Query User Request

Request:

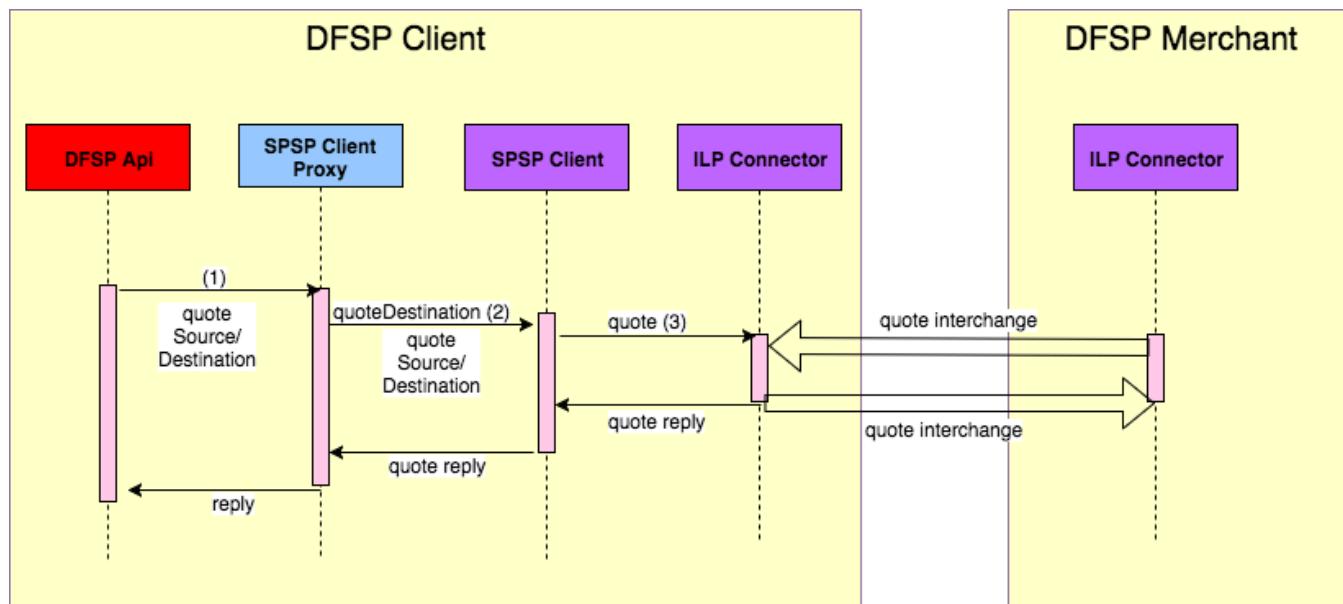
```
GET http://dfsp2-api:8010/v1/receivers/78956562
```

Response:

200 OK

```
{
  "type": "payee",
  "name": "bob",
  "account": "levelone.dfsp2.bob",
  "currencyCode": "USD",
  "currencySymbol": "$",
  "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/
profile_pic.jpg"
}
```

II. Quote Source/Destination



DFSP API -> SPSP CLIENT Proxy

2.1 quoteDestination Method

200 OK

Request:

```
GET http://dfsp1.spss-client/quoteDestinationAmount/  
  
{  
  "receiver": "http://ilp-spss-server:3043/v1/receivers/16023825",  
  "destinationAmount": "10"  
}
```

Response:

```
{  
  "sourceAmount": "10"  
}
```

SPSP CLIENT Proxy -> SPSP CLIENT

2.2 quoteDestination Method

-- to be filled in

SPSP Client -> ILP Connector

2.4 Create Invoice Notification Method

This method will be used for communication between SPSP Client and ILP Connector components.

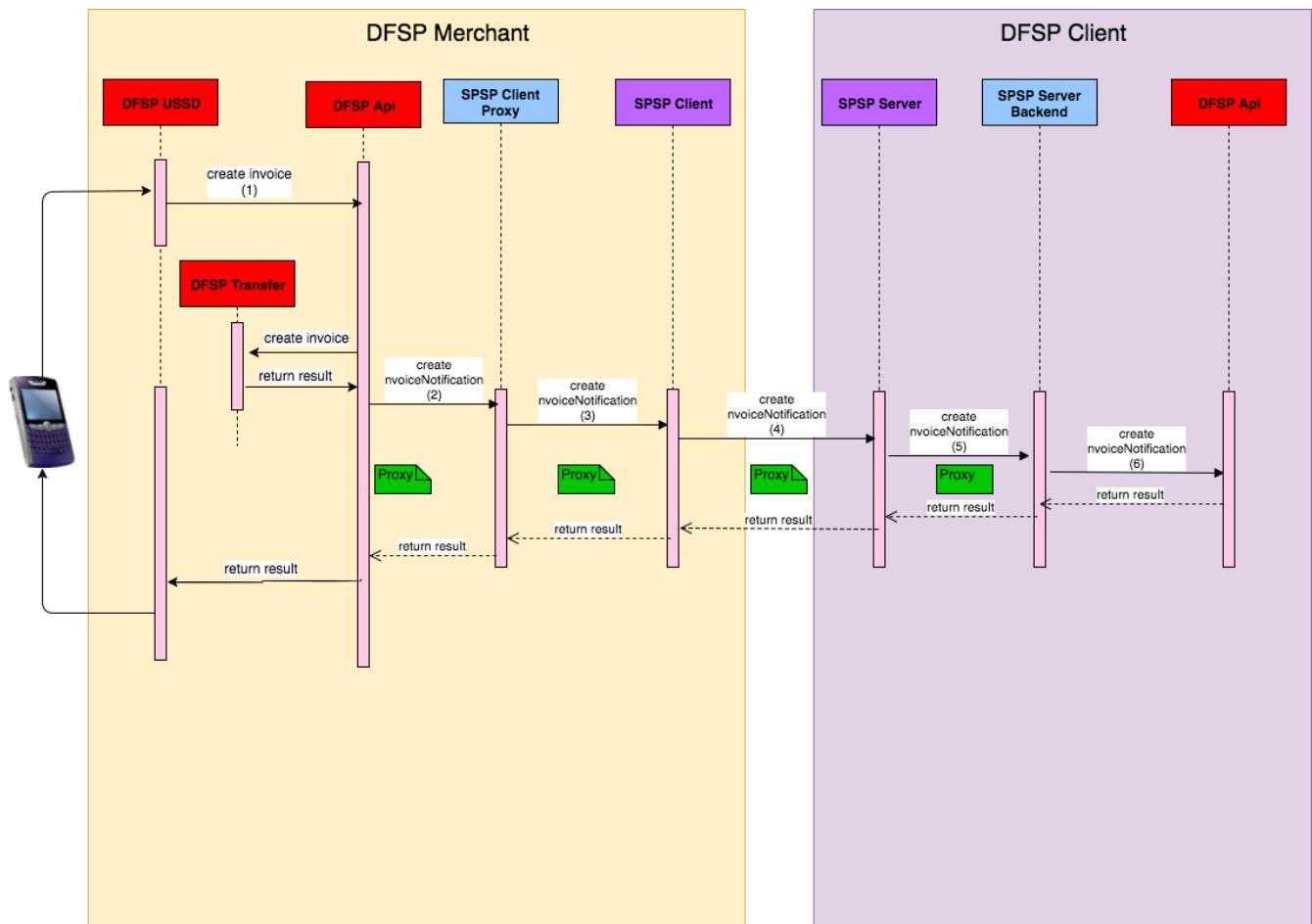
Request:

-- to be filled in

Response:

-- to be filled in

III. Invoice Creation



DFSP USSD -> DFSP API

3.1 Create Invoice Method

Request:

```
POST http://dfsp-api/merchantInvoice
{
  account: 'merchant.account',
  name: 'Merchant name',
  currencyCode: 'USD',
  currencySymbol: '$',
  amount: 100,
  userNumber: 'client.user.number',
```

```
        spspServer: 'client.spspServer',
        invoiceInfo: 'invoice info'
    }
```

Response:

201 Created

- account - merchant's ledger account. e.g. http://dfsp-ledger/ledger/accounts/merchant
- userNumber - client's user number. e.g. '26547070'
- spspServer - baseUrl to client's spsp server. e.g. http://sender-spsp-server/v1

DFSP API -> SPSP CLIENT Proxy

3.2 Create Invoice Notification Method

Request:

```
POST http://dfsp2.spsp-client/invoices/
{
    "invoiceId": "12345",
    "submissionUrl": "dfsp1.spsp-server/v1/invoices",
    "senderIdentifier": "client.user.number",
    "memo": "Invoice from merchant for 100 USD"
}
```

Response:

201 Created

- invoiceId - Id of the invoice that has been generated and stored in the merchant's DFSP.
- submissionUrl - URL to client DFSP. Since this message is send from DFSP-Transfer service to SPSP Client Proxy Service, the receiver service has to know which target SPSP server to contact. This information should be stored in the central directory and can be mapped from a user number.
- memo - field that is going to be displayed on the USSD menu under the 'pending transaction' section

SPSP CLIENT Proxy -> SPSP CLIENT

3.3 Create Invoice Notification Method

-- to be filled in

SPSP Client -> SPSP SERVER

3.4 Create Invoice Notification Method

This method will be used for communication between SPSP Client and SPSP Server components.

Request:

```
POST http://dfsp1.spssp-server/receiver/invoice/
{
    "invoiceUrl": "http://dfsp2.spssp-server/invoice/12345",
    "senderIdentifier": "client.user.number",
    "memo": "Invoice from merchant for 100 USD"
}
```

Response:

```
201 Created
```

- invoiceUrl - full URL to the invoice which is generated and stored in the merchant's DFSP.
- memo - field that is going to be displayed on the USSD menu under the 'pending transaction' section

SPSP SERVER -> SPSP SERVER BACKEND

3.5 Create Invoice Notification Method

-- to be filled in

SPSP Server Backend -> DFSP API

3.6 Create Invoice Notification Method

This method will be invoked from SPSP Server and will be used to create invoice reference the 'DFSP Logic'.

Request:

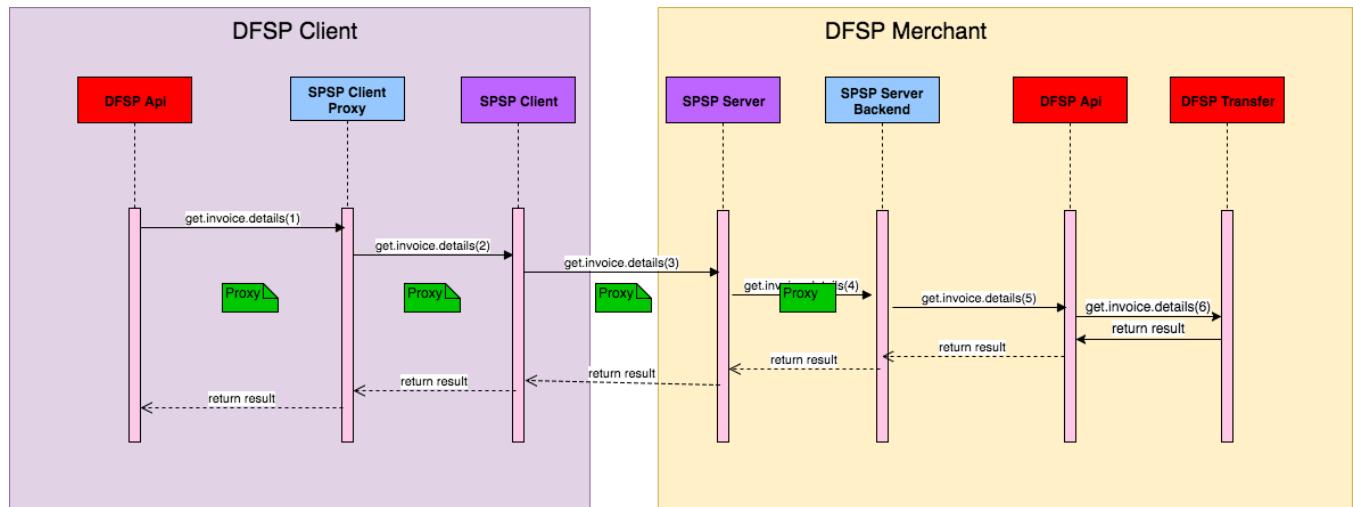
```
POST http://dfsp1.dfsp-api/receiver/invoice/  
  
{  
    "invoiceUrl": "http://dfsp2.spss-server/invoice/12345",  
    "senderIdentifier": "client.user.number",  
    "memo": "Bolagi Shop $5.00"  
}
```

Response:

201 Created

- invoiceUrl - full URL to the invoice which is generated and stored in the merchant's DFSP.
- memo - field that is going to be displayed on the USSD menu under the 'pending transaction section

IV. Get Invoice Details



SPSP CLIENT PROXY / SPSP CLIENT

4.1 Get Invoice Details

Get Invoice details will be done by using the already defined method [GET /v1/query API](#).

Request:

```
GET http://dfsp1.spss-client/invoice?invoiceUrl=http://dfsp2.spss-server/invoice/12345
```

Response:

200 OK

```
{
  "account": "dfsp2.bob.dylan.account",
  "name": "Bob Dylan",
  "currencyCode": "USD",
  "currencySymbol": "$",
  "amount": "10.40",
  "fee": "2.4",
  "status": "unpaid",
  "invoiceInfo": "https://www.example.com/gp/your-account/order-details?ie=UTF8&orderID=111-7777777-1111111"
}
```

The following changes will be introduced:

- Add the invoiceUrl as request parameter, since this API is used between DFSP-Transfer and SPSP Client component. After that the SPSP client component has to have the URL to the merchant DFSP where the invoice is stored.
- type field is removed. No need of such field, because the request is for invoice.
- Add the field 'name' - this is the name of the merchant that has issued the invoice.

SPSP SERVER

4.2 Get Invoice Details

Get Invoice details in SPSP server will be done by using the already defined method [GET invoice](#).

Request:

```
GET http://dfsp2.spss-server/invoice/12345
```

Response:

200 OK

```
{
  "account": "dfsp2.bob.dylan.account",
  "name": "Bob Dylan",
  "currencyCode": "USD",
  "currencySymbol": "$",
  "amount": "10.40",
  "status": "unpaid",
  "invoiceInfo": "https://merchant-website.example/gp/your-account/order-details?ie=UTF8&orderID=111-7777777-111111"
}
```

The following changes will be introduced:

- type field is removed. No need of such field, because the request is for invoice.
- Add the field 'name' - this is the name of the merchant that has issued the invoice.
- Remove field paymentURL - we already have an account field

SPSP Server Backend / DFSP API

4.3 Get Invoice Details

The following new method will be implemented in DFSP API. SPSP Server will call this new method to obtain information about an invoice from the 'DFSP Logic'.

Request:

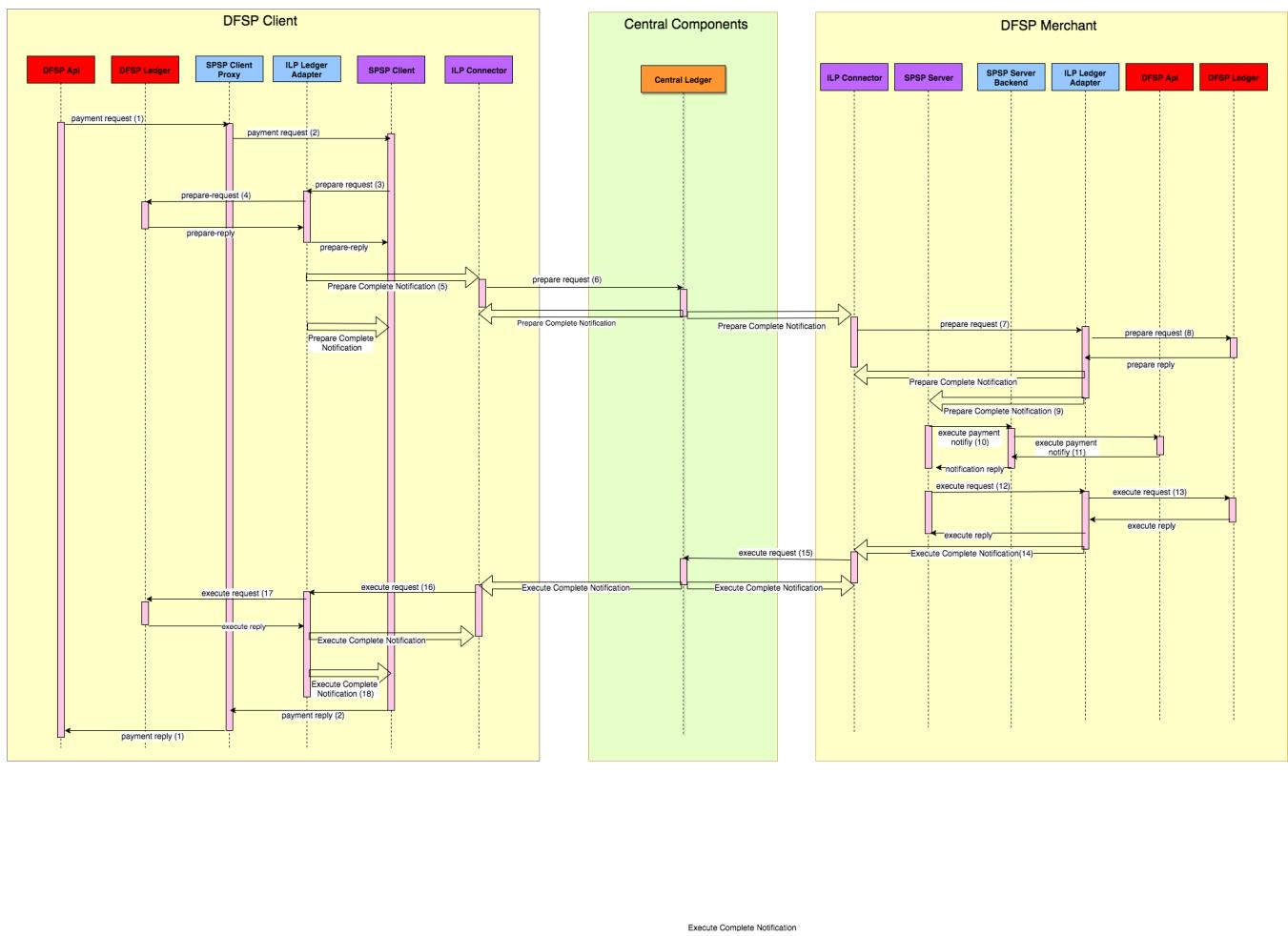
```
GET http://dfsp2.dfsp-api/invoice/12345
```

Response:

200 OK

```
{
  "account": "dfsp2.bob_dylan.account",
  "name": "Bob Dylan",
  "currencyCode": "USD",
  "currencySymbol": "$",
  "amount": "10.40",
  "status": "unpaid",
  "invoiceInfo": "https://merchant-website.example/gp/your-account/order-details?ie=UTF8&orderID=111-7777777-111111"
}
```

V. Invoice Payment



DFSP API -> SPSP CLIENT Proxy

5.1 Invoice Payment Method

Request:

```
PUT  http://dfsp1.spss-client-proxy/spss/client/v1/payments/{uuid}

{
    "sourceIdentifier": "65144444",
    "sourceAccount": "http://dfsp1-ledger/ledger/accounts/bob",
    "receiver": "http://ilp-spss-server/v1/receivers/92806391",
    "destinationAmount": "17",
```

```

    "currency": "USD",
    "fee": 0,
    "memo": {
        "fee": 0,
        "transferCode": "invoice",
        "debitName": "bob dylan",
        "creditName": "alice cooper"
    }
}

```

Response:

200 OK

```

{
    "receiver": "http://ilp-spsp-server/v1/receivers/92806391",
    "sourceAccount": "http://spsp/ilp/ledger/v1/accounts/bob",
    "destinationAmount": "17.00",
    "memo": {
        "fee": 0,
        "transferCode": "invoice",
        "debitName": "bob dylan",
        "creditName": "alice cooper",
        "debitIdentifier": "65144444",
        "sourceIdentifier": "65144444",
        "sourceAmount": "17.00",
        "fulfillment": "hi8Rtk8WiOQkww5bpeXrSoRj41bPXR8c3hfn5i_6zyQ",
        "status": "executed"
    }
}

```

SPSP CLIENT PROXY -> SPSP CLIENT

5.2 Invoice Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

SPSP CLIENT -> ILP LEDGER ADAPTER

5.3 Prepare Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP LEDGER ADAPTER -> DFSP Ledger

5.4 Prepare Payment

PUT http://dfsp-ledger/ledger/transfers/{uuid}

Request:

```
{
  "id": "78311ff6-377e-4f18-8ad2-a919d4b735b1",
  "ledger": "http://spsp-server-backend/ilp/ledger/v1",
  "debits": [
    {
      "account": "http://spsp-server-backend/ilp/ledger/v1/accounts/dfsp1-testconnector",
      "amount": 10,
      "authorized": true,
      "memo": {
        "source_transfer_ledger": "levelone.ist.",
        "source_transfer_id": "7df88b8a-971e-4c95-8402-35196f16855a",
        "source_transfer_amount": "1000"
      }
    }
  ],
  "credits": [
    {
      "account": "http://spsp-server-backend/ilp/ledger/v1/accounts/PerfTest3",
      "amount": 10,
      "memo": {
        "ilp": "AYIBhgAAAAAAAPo0mx1dmVsb251LmRmc3AxL1B1cmZUZXN0My50dW1sRldUajNOOE<...>
      }
    }
  ]
}
```

```

R1Yml0SWR1bnRpZmllclwiOlwiMjk5OTk4MDFcIn0iAA"
    }
}
],
"execution_condition":
"ni:///sha-256;ajsbKuxey2ZYhT9LUzUcz1ccczwIhoYfgrGbfuzy-
NA?fpt=preimage-sha-256&cost=32",
"expires_at": "2017-04-28T15:42:55.294Z"
}

```

Response:

200 OK

```

{
  "id": "f251d2b2-d619-4c22-adaa-d0cc1c585999",
  "ledger": "http://spsp-server-backend/ilp/ledger/v1",
  "debits": [
    {
      "account": "http://spsp-server-backend/ilp/ledger/v1/
accounts/bob",
      "amount": 11,
      "authorized": true
    }
  ],
  "credits": [
    {
      "account": "http://spsp-server-backend/ilp/ledger/v1/
accounts/dfsp2-testconnector",
      "amount": 11,
      "memo": {
        "ilp": "AYIBgQAAAAAAAARMNGxldmVsb251LmRmc3AxLm11ci5UQzFROHVYU11OOFpSNnJxc29y
QXo0VExCWEwdGFxSGeCAUBQU0svMS4wCk5vbmn1OibrQVJ5eEdmdkpIewVEWG92RVBEanB3CkVuY3J5cHRpb246IG5vbmUKUGF5bWVudC1JZDogZjI1MWQyYjItZDYxOS00YzIyLWFkYWEtZDBjYzFjNTg1OTk5CgpDb250ZW50LUxlbd0aDogMTM1CkNvbnR1bnQtVHlwZTo
gYXBwbGljYXRpb24vanNvbgtZw5kZXItSWR1bnRpZmllcjogOTI4MDYzOTEKCiJ7XCJm
ZWVcIjowLFwidHJhbnNmZXJDb2R1XCI6XCJpbnZvaWN1XCIsXCJkZWJpdE5hbWVcIjpcImFsawN1IGNvb3BlclwiLFwiY3J1ZG10TmFtZVwiOlwibWVYIGNoYW50XCIsXCJkZWJpdE
1kZW50aWZpZXJcIjpcIjkyODA2MzkxXCJ9IgA"
    }
  ],
  "execution_condition":
"ni:///sha-256;xQxWWKWXGw7JaYI62rZ5uu3mXLAnjf2yJbrtC3Xv4Sk?fpt=preima
ge-sha-256&cost=32",
"expires_at": "2017-04-28T15:55:16.421Z"
}

```

```
}
```

ILP LEDGER ADAPTER -> ILP CONNECTOR

5.5 Prepare Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP CONNECTOR -> CENTRAL LEDGER

5.6 Prepare Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP CONNECTOR -> ILP LEDGER ADAPTER

5.7 Prepare Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP LEDGER ADAPTER -> DFSP LEDGER

5.8 Prepare Payment

Request:

```
PUT http://dfsp-ledger/ledger/transfers/{uuid}
{
    "uuid": "2527962e-5bbd-460e-8945-154a34f17dba",
    "debitAccount": "bob",
    "debitMemo": {},
    "creditAccount": "dfsp1-testconnector",
    "creditMemo": {
        "ilp_header": {
            "account": "levelone.dfsp2.alice.~psk.UZVSWwsy6ww.MFsvYpK_6SY5BpNLznWs8g.2527962e-5bbd-460e-8945-154a34f17dba",
            "amount": "2.00",
            "data": {
                "data": {
                    "memo": "{\"fee\":0,\"transferCode\":\"invoice\",\"debitName\":\"bob dylan\",\"creditName\":\"alice cooper\"}",
                    "senderIdentifier": "00427080"
                },
                "expires_at": "2017-04-25T17:32:48.384Z"
            }
        }
    },
    "amount": 2,
    "executionCondition": "ni:///sha-256;C2EmAmpD_dylQ8iii4S-afyvxINo-TRomDJI5tgc-0Y?fpt=preimage-sha-256&cost=32",
    "authorized": true,
    "expiresAt": "2017-04-25T17:32:48.384Z"
}
```

Response:

200 OK

```
{
    "id": "http://spsp/ledger/transfers/2527962e-5bbd-460e-8945-154a34f17dba",
    "ledger": "http://spsp/ledger",
    "debits": [
        {
            "account": "http://spsp/ledger/accounts/bob",
            "memo": {},
            "amount": "2.00",
            "authorized": true
        }
    ],
    "credits": []
}
```

```

    "account": "http://spsp/ledger/accounts/dfsp1-testconnector",
    "memo": {
        "ilp_header": {
            "account": "levelone.dfsp2.alice.~psk.UZVSWwsy6ww.MFsvYpK_6SY5BpNLznWs8g.2527962e-5bbd-460e-8945-154a34f17dba",
            "amount": "2.00",
            "data": {
                "data": {
                    "memo": "{\"fee\":0,\"transferCode\":\"invoice\",\"debitName\":\"bob dylan\",\"creditName\":\"alice cooper\"}",
                    "senderIdentifier": "00427080"
                },
                "expires_at": "2017-04-25T17:32:48.384Z"
            }
        },
        "amount": "2.00"
    ],
    "execution_condition": "ni:///sha-256;C2EmAmpD_dylQ8iiI4S-afyvxINo-TRomDJI5tgc-0Y?fpt=preimage-sha-256&cost=32",
    "cancellation_condition": null,
    "state": "prepared",
    "expires_at": "2017-04-25T17:32:48.384Z"
}
}

```

ILP LEDGER ADAPTER -> SPSP SERVER

5.9 Prepare Payment

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

SPSP SERVER -> SPSP SERVER BACKEND

5.10 Execute Payment Notify

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

SPSP SERVER BACKEND -> DFSP API

5.11 Execute Payment Notify

Request:

```
PUT http://dfsp-api/receivers/  
invoices/{invoiceId}/payments/{paymentid}  
  
{  
    "destinationAmount": "1200",  
    "memo":  
        "\"{\\\\\"fee\\\\\"}:0,\\\\\"transferCode\\\\\":\\\\\\\"invoice\\\\\",\\\\\"debitName\\\\\":  
        \"\\\\\"alice cooper\\\\\",\\\\\"creditName\\\\\":\\\\\\\"merchant\\\\\",\\\\\"debitIdentifier\\\\\":  
        \\\\\"92806391\\\\\"}\\\\\"",  
    "status": "proposed",  
    "transferId": "883bb6ba-f425-4bad-80d0-8588f0c192de",  
    "invoiceId": "16",  
    "paymentid": "ce8dfeel-f6be-4b89-a44b-06fb941e0b2"  
  
}
```

Response:

```
200 OK {}
```

SPSP SERVER -> ILP-LEDGER ADAPTER

5.12 Execute Payment Request

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP-LEDGER ADAPTER -> DFSP LEDGER

5.13 Execute Payment Request

Request:

```
PUT http://dfsp-ledger/ledger/transfers/{transferId}/fulfillment

{
  "transferId": "6a4c78f4-cc2f-488f-b04d-26b71c92962a",
  "fulfillment": "oCKAIIXvEbZPFojkJML-W6Xl60qEY-NWz10fHN4X5-Yv-s8k",
  "condition": "ni:///sha-256;sbawjV8idAMItrwviBMq4zMOKuo_1RLNMm1KPPVFM2A?fpt=preimage-sha-256&cost=32"
}
```

Response:

Returns transfer fulfillment.

200 OK

"oCKAIIXvEbZPFojkJML-W6Xl60qEY-NWz10fHN4X5-Yv-s8k"

ILP-LEDGER ADAPTER -> ILP-CONNECTOR

5.14 Execute Payment Notification

Request:

-- to be filled in

Response:

-- to be filled in

ILP-CONNECTOR -> CENTRAL LEDGER

5.15 Execute Payment Request

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP-CONNECTOR -> ILP LEDGER ADAPTER

5.16 Execute Payment Request

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

ILP LEDGER ADAPTER -> DFSP LEDGER

5.17 Execute Payment Request

Request:

```
PUT http://dfsp-ledger/ledger/transfers/{transferId}/fulfillment

{
    "transferId": "6a4c78f4-cc2f-488f-b04d-26b71c92962a",
    "fulfillment": "oCKAIYvEbZPFojkJML-W6X160qEY-NWz10fHN4X5-Yv-
s8k",
    "condition":
"ni:///sha-256;sbawjV8idAMItrwviBMq4zMOKuo_1RLNMm1KPPVFM2A?fpt=preima
ge-sha-256&cost=32"
}
```

Response:

Returns transfer fulfillment.

200 OK

```
"oCKAIYvEbZPFojkJML-W6X160qEY-NWz10fHN4X5-Yv-s8k"
```

ILP LEDGER ADAPTER -> SPSP CLIENT

5.18 Execute Payment Notification

Request:

```
-- to be filled in
```

Response:

```
-- to be filled in
```

Pending Transfers API

I. Summary

Creating and paying or rejecting pending transfers using the DFSP Over the Top API.

In real life, this flow can be seen for merchant purchase use case where a merchant sells goods to customer and both of them have mobile (smart) phones. From end user's perspective the use case looks like this:

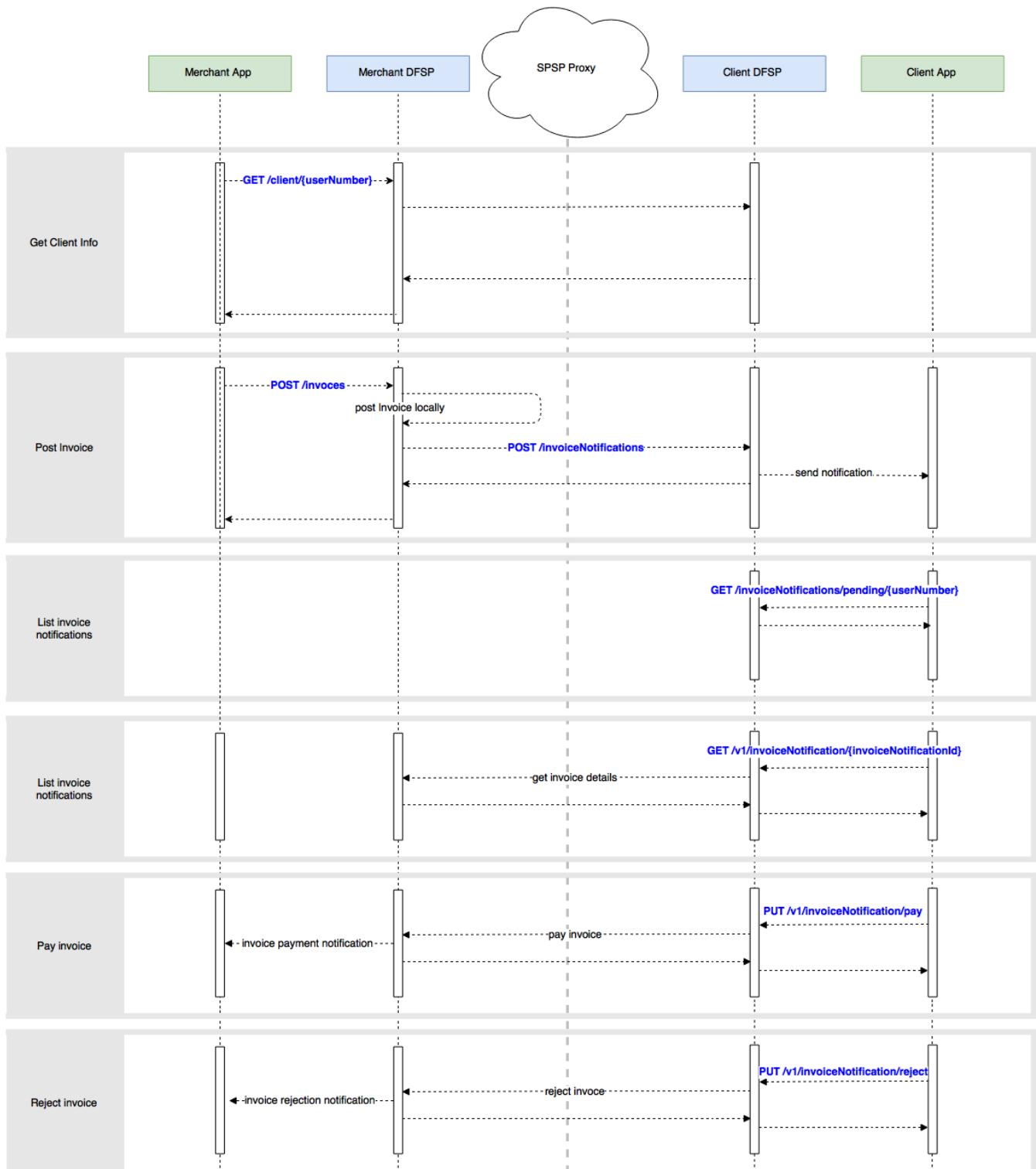
1. Merchant log in his smart application and select 'merchant purchase' option.
2. Merchant enters the customer identification (phone number or user number) or selects it from a list (for regular customers visiting his shop).
3. The Merchant enters the amount that the customer has to pay and confirms the operation
4. The system creates an 'invoice' for that amount into merchant's DFSP and send a notification to the customer that he has a pending invoice.
5. The customer log in his smart application and see the pending notification that he can pay or reject. He also can go to pending invoices menu where he can see a list of all the pending invoices for him.
6. The customer selects an invoice that he is going to pay and the system loads all the details for the invoice, together with the fee associated with the transaction.
7. The customer can pay or reject this pending invoice. Upon paying the customer has to authenticate the transaction (i.e in case the application is configured to use 2 factor authentication), the customer initiate the payment and the merchant gets notification that the invoice has been payed successfully.

The API exposes the DFSP functionalities for transfers processing, customer and account management, etc. to third party applications such as Android/IPhone smart application. There are additional set of APIs which are used for communication between DFSP to DFSP. Those APIs will not be analyzed in the current document.

API Principles

- Restful approach to API design.

- Based on JSON, no other content types are supported.



II. Assumptions

- Merchant is the party sending the invoices.
- Client is the party who is receiving the invoices and is able to pay/reject them.
- The merchant and the client are in different systems.
- The merchant and the client are logged in to their systems.
- Security is not a part of this specification.
- Getting information about the customer accounts is not a part of this specification.
- Notifications are not a part of this specification, as each platform provides different means for that (e.g. Google Firebase notification services, SMS notification, email notification, etc.).

III. Get Client Information

This API will return the information about a client (his first name, last name and a photo) based on client identification such as user number or phone number.

API Description

- **URL**

/v1/client/{userNumber}

- **Method**

GET

- **URL Params**

- userNumber - The number of the user

- **Sample Call**

```
curl -X GET --header 'Accept: application/json' 'http://host/v1/client/78956562'
```

- **Success Response**

- **Code:** 200

Content

- firstName [string] - Client's first name
- lastName [string] - Client's last name
- imageUrl [string] - Link to the client's image

- **Sample Response**

```
{ "firstName": "Bob", "lastName": "Smith", "imageUrl": "https://red.ilpdemo.org/api/receivers/bob?securityKey=497a2af553cc1cc6443692c6eed60ebda1a08020d287c4f5461ale41d840d6f5"} * Error handling
```

- **Code:** 401

Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }

- **Code:** 404

Content { "id": "ClientNotFound", "message": "Client userNumber could not be found" }

IV. Create Invoice

This API will create an invoice in the merchant's DFSP and will send notification to the client's DFSP. The invoice is created into the merchant's DFSP associated with a merchant's account. Thus then the invoice is paid the money will go into the associated account. The client's DFSP upon receiving the invoice notification can send it to the third party application via SMS, Google Notification service, etc. It is recommended that the invoice reference in the client's DFSP will not be associated with any clients account thus the client can choose an account from which he is going to pay the invoice.

API Description

- **URL**

/v1/invoices

- **Method**

POST

- **Data Params**

Required

- account [string] - Invoice merchant's account
- amount [number] - Invoice amount
- userNumber [string] - Client's user number

Optional

- info [string] - Additional invoice information

- **Sample Call**

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "account": "merchant", "amount": 123, "userNumber": "78956562", "info": "Invoice from merchant to Alice" }' 'http://host/v1/invoices'
```

- **Success Response**

- **Code:** 200
Content

- invoiceId [number] - Invoice id
- account [string] - Invoice merchant's account
- firstName [string] - Merchant's first name
- lastName [string] - Merchant's last name
- currencyCode [string] - Invoice merchant's currency code
- currencySymbol [string] - Invoice merchant's currency symbol
- amount [string] - Invoice amount
- status [string] - Invoice status
- userNumber [string] - Invoice client's user number
- info [string] - Invoice additional information

- **Note** Invoice status can be one of the following:

- paid
- pending
- rejected

- **Sample Response**

```
{ "invoiceId": 1, "account": "merchant", "firstName": "John", "lastName": "Smith", "currencyCode": "USD", "currencySymbol": "$", "amount": "130.34", "status": "pending", "userNumber": "78956562", "info": "Invoice from merchant for 130.34 USD" } * Error handling
```

- **Code:** 401

```
Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }
```

- **Code:** 404

```
Content { "id": "ClientNotFound", "message": "Client userNumber could not be found" }
```

- **Code:** 404

```
Content { "id": "MerchantNotFound", "message": "Merchant account could not be found" }
```

V. Pending Invoice Notifications

Client will be able to obtain a list of all the pending invoices. This API is optional for the use case. Depending on the implementation it can be useful in case the third party application implements an option for the client to list all the pending invoices that the client has.

API Description

- **URL**

/v1/invoiceNotifications/pending/{userNumber}

- **Method*

GET

- **URL Params**

- userNumber - Client's user number

- **Sample Call**

```
curl -X GET --header 'Accept: application/json' 'http://host/v1/invoiceNotifications/pending/78956562'
```

- **Success Response**

- **Code:** 200

Content

- invoiceNotificationId [number] - Invoice notification id
- status [string] - Invoice notification status
- info [string] - Additional invoice notification information

- **Note** Invoice notification status can be one of the following:

- paid
- pending
- rejected

- **Sample Response**

```
{ "invoices": [ { "invoiceNotificationId": 2, "status": "pending", "info": "Invoice from merchant for 130.34 USD" } ] }
```

- **Error handling**

- **Code: 401**

Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }

- **Code: 404**

Content { "id": "ClientNotFound", "message": "Client userNumber could not be found" }

- [Try it out here](#)

VI. Get Payment Details by Invoice Notification

This API will return to the client all the details associated with the payment for a certain invoice such as the details about the merchant (first name, last name) and the fee associated with the transaction.

API Description

- **URL**

```
/v1/invoicesNotifications/{invoiceNotificationId}
```

- **Method**

GET

- **URL Params**

- invoiceNotificationId - Invoice notification id

- **Sample Call**

```
curl -X GET --header 'Accept: application/json' 'http://host/v1/invoiceNotifications/2'
```

- **Success Response**

- **Code: 200**

- **Content**

- firstName [string] - Merchant's first name
 - lastName [string] - Merchant's last name
 - amount [number] - Invoice amount
 - currencyCode [string] - Currency code
 - currencySymbol [string] - Currency symbol
 - fee [number] - Transfer fee

- **Sample Response**

```
{ "firstName": "Ben", "lastName": "Smith", "amount": 123, "currencyCode": "USD", "currencySymbol": "$", "fee": 1.23, }
```

- **Error handling**
- **Code: 401**
Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }
- **Code: 404**
Content { "id": "InvoiceNotificationNotFound", "message": "Invoice notification with such invoiceNotificationId could not be found" }
- **Try it out here**

VII. Pay Invoice

This API will be used by the client's application to request a payment for the invoice. Upon successful payment, merchant's DFSP will mark the invoice as paid and the merchants application should get an invoice payment notification (outside the scope of the current document).

API Description

- **URL**

/v1/invoiceNotifications/pay

- **Method**

PUT

- **Data Params**

Required

- account [string] - Client's account
- invoiceNotificationId [string] - Invoice notification id

- **Sample Call**

```
curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "account": "bob", "invoiceNotificationId": "6" }' 'http://host/v1/invoiceNotifications/pay'
```

- **Success Response**

- **Code:** 200
Content

- invoiceNotificationId [string] - Invoice notification id
- status [string] - Invoice notification status

- **Note** Invoice notification status can be one of the following:

- paid
- pending
- rejected

- **Sample Response**

```
{ "invoiceNotificationId": "3", "status": "paid", }
```

- **Error handling**
- **Code: 400**
Content { "id": "InvoiceNotificationNotFound", "message": "Invoice notification with such invoiceNotificationId could not be found" }
- **Code: 401**
Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }
- **Code: 500**
Content { "id": "InsufficientFunds", "message": "You do not have sufficient funds to pay the invoice" }
- **Try it out here**

VIII. Reject Invoice

Client also will be able to reject invoices. In most of the cases this API will be used to instruct the client's DFSP to remove this invoice notification from the client's list and to send the reject notification to the merchant's DFSP.

API Description

- **URL**

/v1/invoiceNotifications/reject

- **Method**

PUT

- **Data Params**

Required

- invoiceNotificationId [string] - Invoice notification id

- **Sample Call**

```
curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "invoiceNotificationId": "2" }' 'http://host/v1/invoiceNotifications/reject'
```

- **Success Response**

- **Code:** 200

- Content

- invoiceNotificationId [string] - Invoice notification id
 - status [string] - Invoice notification status

- **Note** Invoice notification status can be one of the following:

- paid
 - pending
 - rejected

- **Sample Response**

```
{ "invoiceNotificationId": "2", "status": "rejected", }
```

- **Error handling**

- **Code 400**

Content { "id": "InvoiceNotificationNotFound", "message": "Invoice notification with such invoiceNotificationId could not be found" }

- **Code: 401**

Content { "id": "Unauthorized", "message": "Access is denied due to invalid credentials" }

- **Try it out here**

Bulk Payments

Bulk Payment Initiation

In the DFSP there should be a web interface available where every user (with certain user rights) of the DFSP can login with user number and PIN and initiate a bulk payment.

When initiating a bulk payment the user should be able to select an account from which the bulk payments will be send.

L1P shall support upload of the bulk file with payments from a web interface. The file format should be .csv. The fields in the file will be:

- Sequence Number (row)
- Identifier
- First Name
- Last Name
- Date of birth
- National id
- Amount

Maker/Checker

There is a maker/checker concept implemented for bulk payments. Maker and Checker are roles that can be assigned to different users. Maker role is going to upload the file with the bulk payments and checker role is going to initiate the bulk payments.

Bulk File Verification

All the fields in the bulk file shall be mandatory.

In case the some of the required fields are not valid the funds will not be disbursed to this user.

Handling Fees

The discriminatory fees should be recorded in the same way as it is done for a regular transaction - different row in the ledger for each payment. There is possibility to configure different fees per different transaction type, so there could be special set of fee configured for the bulk payments. The rational behind having the fees recorded as a separate line for each transaction is:

- Calculation and record of the discriminatory fees are done within the DFSP and we do not foresee any performance issues to come from that approach.
- The fees must be recorded as a separate row in the ledger

Re-sending of Funds in Case a Temporary Error is Detected

Upon sending bulk payments the system should detect temporary errors and retry sending of funds again to those user.

Temporary errors could be technical such as missing connectivity or some service is down or non technical such as not enough funds in the account, tier limit is hit, etc.

The system should detect a special case - a user has already a user number but for a service different from mWallet/bank account. In this case the system should notify the user to open a mWallet/bank account.

When initiating a bulk payment the system shall allow entering of the end date/time after which the bulk report will be

finalized and no more retries to recover from temporary errors will be done.

The L1P should have some logic for retrying bulk payments which had failed due to temporary errors. There is a configuration on DFSP level for the minimum retry time for bulk payments.

Out-of Scope Scenarios

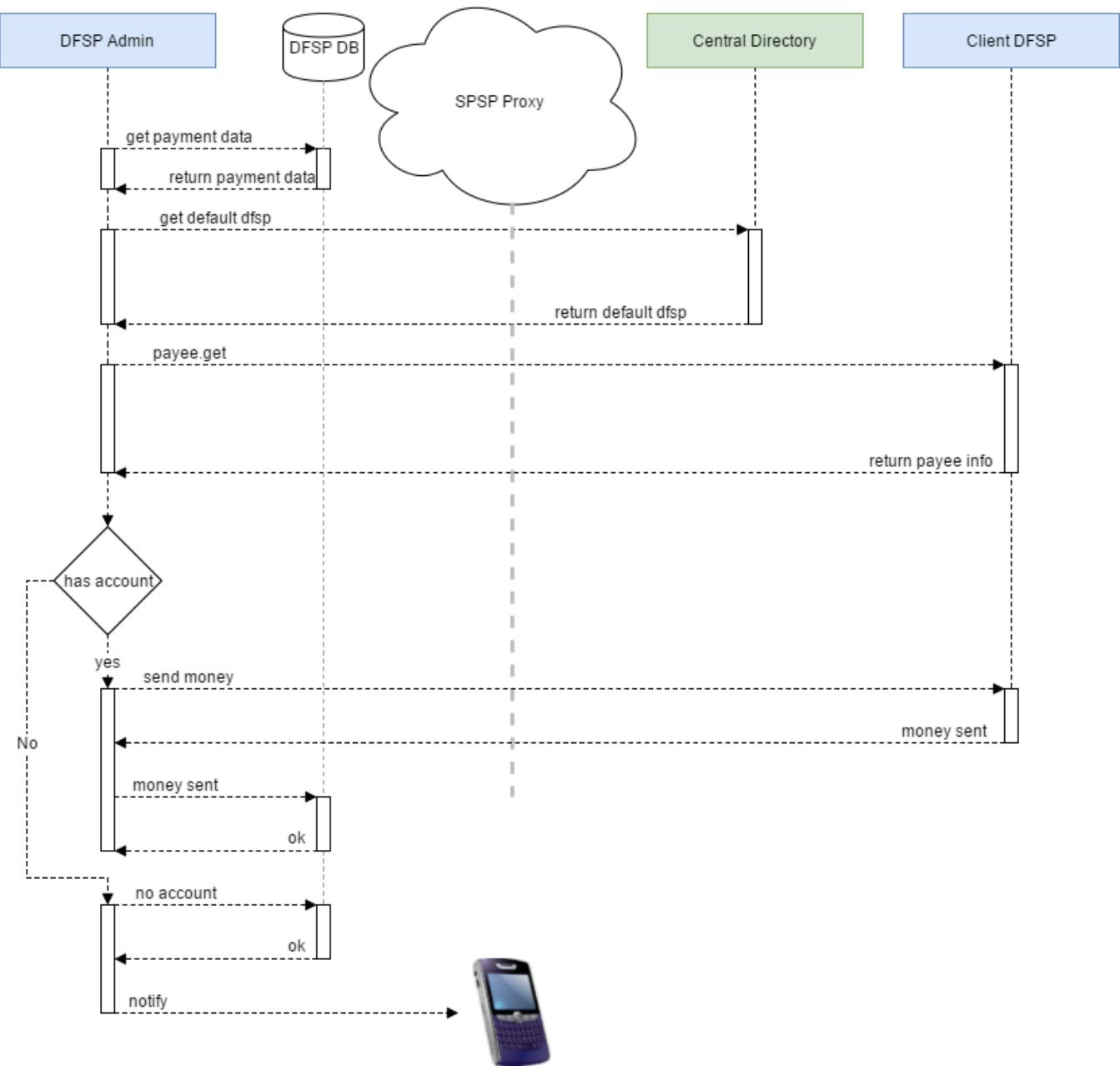
The L1P shall reject component transactions where named participants are ineligible to transact.

Rationale: Payments must be subject to fraud and AML/CFT checks, and blocked where the participant is deemed an unacceptable recipient. www.gatesfoundation.org Requirements for a Pro-Poor Interoperability Service for Transfers.

Currently we don't have such components in the system to support this scenario.

Handling Bulk Payments Transactions

The diagram below illustrates both cases - whether the user has a mwallet account or not. *In case the user has an existing mwallet account the money will be attempted to be transferred to it.* In the other case - the user's default DFSP will indicate that there's no mwallet account associated with the given user and therefore a temporary error will be recorded in sending DFSP's database and the user will be notified.



The following changes have to be implemented:

Central directory get user API

Current Request:

```
GET http://ec2-35-163-231-111.us-west-2.compute.amazonaws.com:8088/
directory/v1/resources?identifierType=eur&identifier=26547070
```

Current Response:

```
{
  "spspReceiver": "http://ec2-35-163-231-111.us-
```

```
west-2.compute.amazonaws.com:3043/v1"  
}
```

- To be able to query for many users we should enable posting of multiple 'identifierTypes' and 'identifiers'.
- To support posting to a user number which does not have mWallet/bank account opened, we need to define another 'identifierType' e.g. phone subscribers and to be able to do a query of if. The response shouldn't be a spspServer, but a 'notificationURL'.

SPSP Protocol

- Get Payee Details (changes in this API is needed only if we have to compare names in DFSP against names from the uploaded bulk file).

Current Request:

```
GET http://ec2-35-163-231-111.us-west-2.compute.amazonaws.com:8088/  
spsp/client/v1/query?receiver=http://ec2-35-163-231-111.us-  
west-2.compute.amazonaws.com:3043/v1/receivers/26547070
```

Current Response:

```
{  
  "type": "payee",  
  "name": "bob",  
  "account": "levelone.dfspl.bob",  
  "currencyCode": "USD",  
  "currencySymbol": "$",  
  "imageUrl": "https://red.ilpdemo.org/api/receivers/bob/  
profile_pic.jpg"  
}
```

We have to enable API to be able to query for multiple receivers

Quote Destination Amount This API is needed only if we have to get connector fees.

Current Request:

```
GET http://ec2-35-163-231-111.us-west-2.compute.amazonaws.com:8088/  
spsp/client/v1/  
quoteDestinationAmount?receiver=http://ec2-35-163-231-111.us-  
west-2.compute.amazonaws.com:3043/v1/receivers/  
26547070&destinationAmount=32
```

Current Response:

```
{  
  "sourceAmount": "32"  
}
```

This API should be changed to support multiple receivers/amounts and get back the information for them.

Setup/Payment

Current Request:

```
POST http://ec2-35-163-231-111.us-west-2.compute.amazonaws.com:8088/  
spsp/client/v1/setup  
{  
  receiver: "http://ec2-35-163-231-111.us-  
west-2.compute.amazonaws.com:3043/v1/receivers/26547070",  
  sourceAccount: "http://ec2-35-163-231-111.us-  
west-2.compute.amazonaws.com:8014/ledger/accounts/alice",  
  destinationAmount: 33,  
  memo: {  
    fee: 1,  
    transferCode: 'p2p',  
    debitName: 'alice',  
    creditName: 'bob'  
  },  
  sourceIdentifier: '85555384'  
}
```

Current Response:

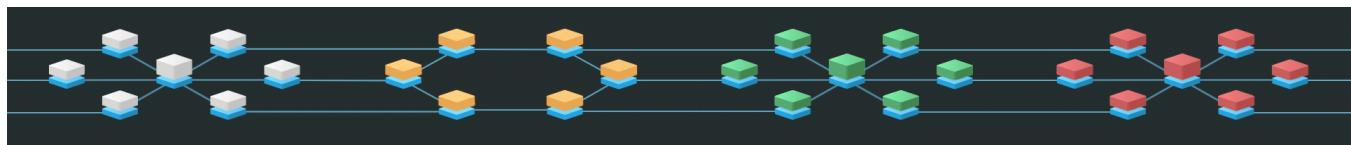
```
{  
  "id": "",  
  "address": "",  
  "destinationAmount": "",  
  "sourceAmount": "",  
  "sourceAccount": "",  
  "expiresAt": "",  
  "data": {  
    "senderIdentifier": ""  
  },  
  "additionalHeaders": "",  
  "condition": ""  
}
```

This API should be changed to support multiple receivers/amounts and get back the information for them.

Interledger for Mojaloop

The Interledger project is a suite of protocol definitions and reference implementations that define a standard way to connect any number of disparate payment systems together into one interconnected network: an *internet of value*.

Mojaloop uses Interledger as its settlement layer so that individual instances or deployments of Mojaloop software can eventually become interconnected not just with one another, but with all other payment systems worldwide. Interledger development is spearheaded by Ripple, with support from the W3C and various other stakeholders.



Interledger provides a standard for linking disparate payment networks to one another.

Contents:

- [Why Interledger](#)
- [Core Concepts](#)
 - [Ledgers](#)
 - [Connectors](#)
 - [Cryptographic Proof](#)
 - [Forward Holds, Backwards Execution](#)
- [Protocol Layers](#)
- [Addresses and Routing](#)
 - [Prefix-Based Routing](#)
- [Data Formats](#)
- [Software Components](#)
 - [Running the ILP Software](#)
 - [Execution Flow](#)

Why Interledger

The Right Technology

Interledger's features and capabilities closely align with Mojaloop's principles, and Interledger is the most advanced standard for interconnectivity at this time.

For the Right Context

The Bill & Melinda Gates Foundation has observed the trends, problems, and opportunities in developing countries in the world and developed principles to guide the direction of a financial system that can benefit the most people:

- A *push payment model* with immediate funds transfer and same-day settlement. Interledger is compatible with a push payment model and can enable settlement within seconds, depending on the limitations of the transacting parties.
- *Open-loop interoperability* between providers. Interledger is intended to be an open standard that *anyone* can build on, enabling innovation and interoperability without the usual boundaries
- Adherence to well-defined and adopted *international standards*. Interledger is being developed by a trans-national community with open standards, in collaboration with the W3C and other standards organizations.
- Meeting or *exceeding the convenience, cost, and utility of cash*. Interledger's open standards, incredibly fast digital settlement, and open connectivity add up to a system that can be far cheaper, faster, and more convenient than cash for a wider range of transaction values.

The two remaining principles for Mojaloop are system-wide *shared fraud and security protection* and *identity and know-your-customer (KYC) requirements*. Interledger does not have specific provisions for either one of these, but it does not preclude participants from building systems that enforce such restrictions. In fact, Interledger has been designed on the assumption that providers of different types and in different contexts must have different restrictions and needs for fraud, security, and identity requirements. Mojaloop is spearheading one of the first sets of fraud detection and information sharing to be built into an Interledger compatible system.

Core Concepts

Ledgers

Interledger conceptualizes a *ledger* as a system tracking accounts and balances in a single currency. In the real world, there are systems called "ledgers" that support multiple currencies; in Interledger parlance, each supported currency in such a system would comprise a separate "ledger". The act of sending money from one user of a given ledger to another user of the same ledger is called a *transfer*. A payment that can be executed by a single transfer within a single ledger does not need or use Interledger.



Connectors

The Interledger project assumes that no one ledger will ever serve the whole world. Aside from the problem of scaling a ledger to serve billions of members of humanity, ledgers have different intrinsic qualities that benefit different parties; different ledgers exist today in part because their customers have not just different but *mutually exclusive* needs and preferences. Still, people would like to be able to pay each other even if they don't use the same ledger:



Payments that cross a ledger boundary are currently hard.

Rather than trying to create one ledger to rule them all, we should make payment systems *interoperable*. We do this by connecting systems to each other, then bridging payments through multiple connectors using *Cryptographic Proof*.



Connectors link ledgers to each other. In the LIP model, all DFSPs connect to a central ledger.

The *Connector* is one of the core pieces of ILP software. Each connector is linked to two or more ledgers where it holds a balance, and it facilitates payments by receiving money in one ledger and paying out money in another ledger. Within a single Level One deployment, we expect that each Digital Financial Services Provider (DFSP) runs a connector pairing their home ledger to the central IST ledger, and all the ledgers are denominated in the same currency. In the greater inter-ledger world, a Connector could link two DFSPs directly, and the ledgers could be denominated in any pair of currencies; the connector sets the rate of exchange between each ledger's native currency.

Cryptographic Proof

In traditional payments, each intermediary must be trustworthy. The more intermediaries, the higher the risk of a transaction failing partway through. Interledger solves this problem with the financial equivalent of a two-phase commit. Each transfer in the payment is locked by a *condition* value and unlocked by a *fulfillment* value that hashes to the condition. With interledger, each party only needs to trust the ledger or intermediary immediately adjacent in the chain, regardless of how long many transfers and intermediaries are involved.

To provide the least risk for users, ledgers should provide conditional hold functionality, such that a cryptographic fulfillment automatically executes a prepared transfer. Mojaloop's example ledgers all implement such systems. Interledger has some other requirements for optimal operation, including authenticated best-effort messaging between users of a ledger. For details of ledger requirements and recommendations, see [IL-RFC-17: Ledger Requirements](#).

Forward Holds, Backwards Execution

The typical execution pattern of an Interledger transfer is forward holds (starting with the transfer from originator) followed by backwards execution (starting with the transfer to the beneficiary). First comes a payment planning step wherein the originator asks the beneficiary for a unique cryptographic condition to which the beneficiary knows the answer. (In Mojaloop, the DFSPs provide this functionality on behalf of their customers.) The originator starts by preparing a conditional transfer to an intermediary's account in the sending DFSP's ledger. This intermediary is a *Connector*. In the Mojaloop model, each DFSP runs a Connector service that links the DFSP's ledger to a central Interoperability Service for Transfer (IST) ledger. The Connector chooses a route and prepares a conditional transfer in a different ledger, to either another intermediary Connector or the final beneficiary. That Connector does the same, until the transfer to the beneficiary is prepared in this manner. All the transfers share the same cryptographic condition, which only the beneficiary can fulfill.

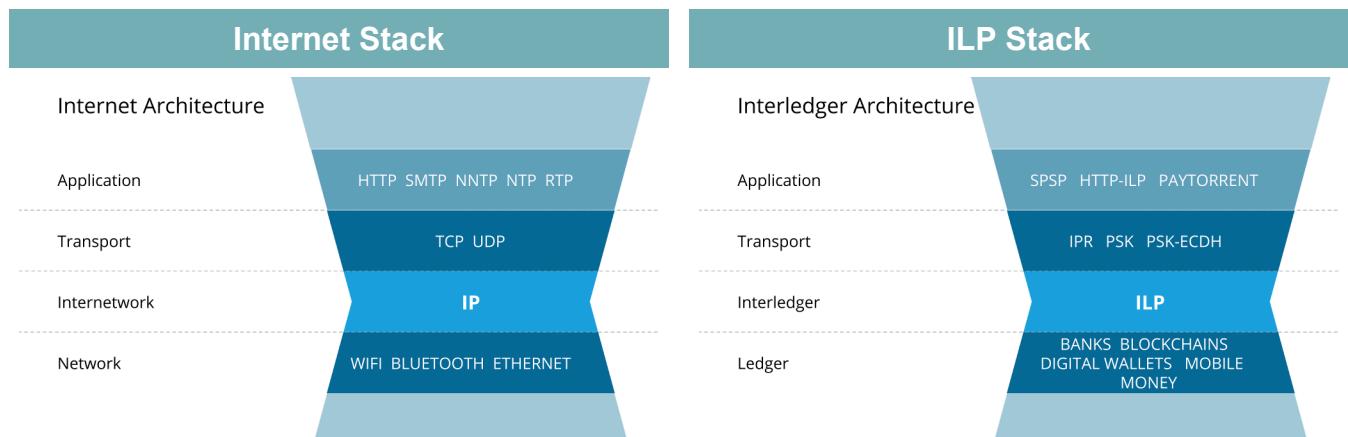
The beneficiary (or, in Mojaloop's case, the beneficiary's DFSP) notices that the incoming transfer with a known cryptographic condition has been prepared in the beneficiary's preferred DFSP ledger. The beneficiary executes this transfer by revealing the cryptographic fulfillment to the DFSP ledger; at this point the beneficiary has gotten paid. The last connector sees the fulfillment in the execution notification, then uses the same fulfillment to execute the previous transfer in the chain. This continues until the first transfer executes, debiting the originator.

Each transfer needs a timeout, or else malicious actors could trick connectors into locking up money forever by proposing transfers that won't be executed. However, with timeouts, it's possible that some transfers of a payment

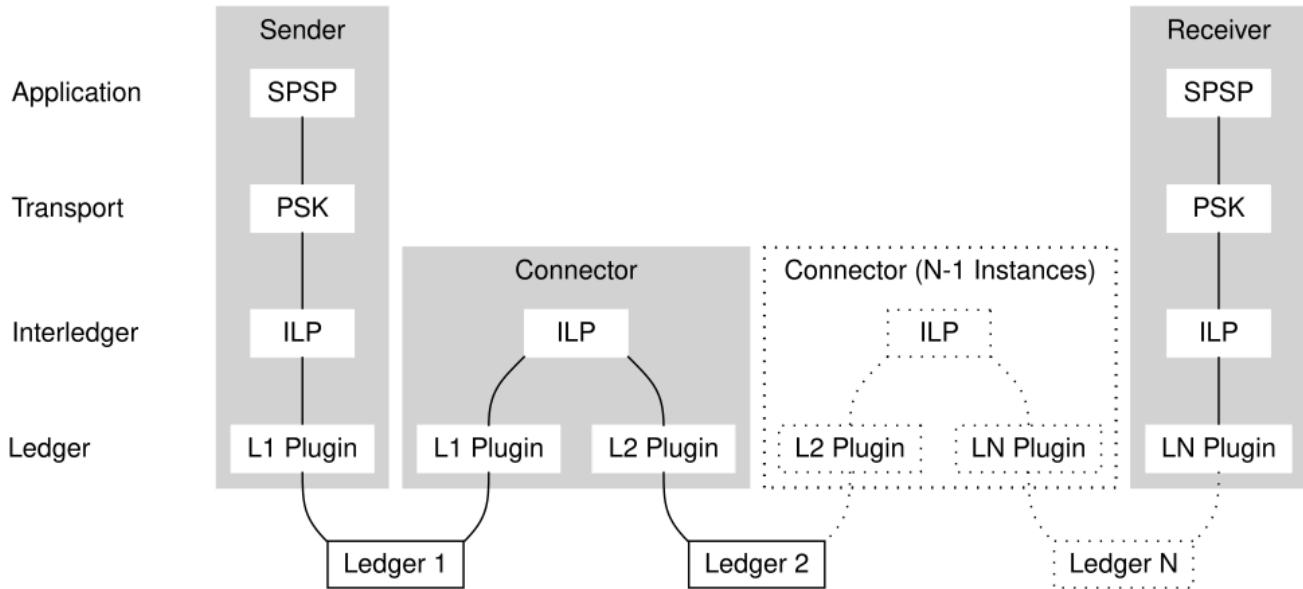
execute in time while other transfers expire, especially if a particular connector or ledger has an outage. The pattern of executing transfers last-to-first protects the customer: it guarantees that the money will be credited to the beneficiary or the originator will not be debited. (In the failure case, both are true.) Connectors in the middle take the risk of losing funds in the failure case, but they can choose the timeouts of each transfer and the fees for exchanging across ledgers in order to minimize the risk. For more information, see [Connector Risks](#).

Protocol Layers

The design of Interledger intentionally copies the design of the Internet as much as is applicable. The four Interledger layers-Ledger, Interledger, Transport, and Application-are analogous to the Data link, Network, Transport, and Application layers of the [OSI model](#). Both models revolve around a single core protocol: Internet Protocol (IP) for the OSI stack, and the Interledger Protocol (ILP) for the Interledger stack.



Another way of looking at the protocol:



The layers of the Interledger protocol stack are as follows:

Layer	Description
<u>Application</u>	Application-defined protocols for planning payment.
<u>Transport</u>	Defines standard data formats for application-layer data and cryptographic condition generation.
<u>Interledger</u>	In this layer, connectors communicate with one another to plan the transfers involved in a payment.
<u>Ledger</u>	In this layer, senders, receivers, and connectors communicate to the ledgers involved in the payment.

Application Layer

The Application Layer coordinates and prepares overall payments. User-facing applications implement protocols from this layer to prepare payments with one another. At this layer, the two endpoints of a payment communicate directly with one another. In Mojaloop, the [Scheme Adapter](#) implements a custom application layer protocol using Interledger Payment Request (IPR) format as the Transport layer. In the L1P's custom protocol, the two DFSP's communicate directly using HTTPS to plan a payment before preparing it.

Transport Layer

The Transport Layer defines how payments are identified and how to generate the cryptographic conditions for the transfers in the payment. Mojaloop uses the IPR format. For the data included in this layer, Mojaloop uses the format defined by the Interledger Pre-Shared Key (PSK) specification, which resembles HTTP headers, although L1P does not use the PSK protocol itself. L1P does not encrypt the data.

Key pieces of data that are defined in this level are:

- The expiration time of the payment
- The key type used to generate the unique condition and fulfillment for this payment
- A unique nonce for the payment
- Mojaloop "Trace ID" of a payment

Note: Unlike the OSI model, the Interledger stack does not have a hard distinction between the "Application" and "Transport" layers; any application layer protocol is closely fixed to a particular transport layer protocol. The main point of the distinction is to make it possible to implement client libraries for transport layer functionality that can be used as generic building blocks for writing application-layer protocols.

More information

- [IL-RFC-11: Interledger Payment Request](#): The IPR specification
- [IL-RFC-16: Pre-Shared Key](#): The PSK specification, which includes the data format recommended for use with IPR

Interledger Layer

There are two closely-related protocols in the Interledger layer: the Interledger Protocol (ILP) and the Interledger Quoting Protocol (ILQP). Connectors communicate with each other in these protocols, using ILQP to quote payments and ILP to prepare payments. (The execution happens individually for each transfer at the ledger layer.)

More information

- [IL-RFC-3: Interledger Protocol](#)
- [IL-RFC-8: Interledger Quoting Protocol](#)

Ledger Layer

The ledger layer is implemented by the unique, core ledgers of each system. In Mojaloop, these ledgers include each DFSP's internal ledger and the IST's central ledger. The core operations are preparing conditional transfers, executing those transfers, and sending authenticated messages to other users of the same ledger.

Each Connector must know how to use the API of the ledgers to which it is connected. Rather than having a unique

API for each ledger, Mojaloop's reference implementations all use a consistent API, called the [Five Bells Ledger API](#). In the case of a DFSP that has an existing ledger API, either the DFSP must run an adapter to provide a Five Bells Ledger API, or the Connector must have a plugin for using the DFSP's own ledger API.

Addresses and Routing

Within the Interledger Protocol (ILP) layer, connectors route payments according to their internal routing tables. The destination of a given ILP payment is determined by its *ILP Address*, a hierarchical string of alphanumeric identifiers analogous to an IP address. For example, all payments to addresses starting with `private.11p.ZZZ.dfsp1.` are routed to the connector operated by DFSP 1.

Currently, Mojaloop participants are not meant to be reachable by the general public, so they use the `private.` prefix.

The details of routing are not specified in the protocol, but the connectors used by Mojaloop follow simple longest-prefix rules with static routing tables.

ILP Addresses are specified by [IL-RFC-15: ILP Addresses](#).

Address Allocation

Each DFSP needs a unique address prefix. The following guidelines establish

1. A standard prefix, `private.11p.`
2. A country or location prefix. In most cases, this can be an [ISO 3166-1 alpha-3 code](#). The code `ZZZ` represents demo instances. If there is any doubt, the operator of the IST chooses which code to use.
3. A unique identifier for the DFSP. The DFSP should suggest a code, and the operator of the IST should confirm that the code is not already in use by another DFSP in the same instance (that is, with the same country/location prefix). Valid characters for the DFSP's segment identifier are alphabetic (A-Z, upper or lower case, case sensitive), digits 0-9, underscore (`_`), tilde (`~`), and dash (`-`). The DFSP identifier should be kept short since the entire address has to fit in 1023 characters, including any sub-account addressing or invoicing information; a good DFSP identifier should be about 20 characters or less.

Example address prefix for "DFSP 1" in a test instance

```
private.11p.ZZZ.dfsp1.
```

The address of a customer account should be the DFSP's prefix and the customer's account number. (It's acceptable to have additional dot-separated segments after the account number to indicate more information about the purpose or destination of a payment.)

Example customer account address for a "medical benefits" sub-account

```
private.11p.ZZZ.dfsp1.849702568.medical
```

Data Formats

Interledger standards are defined using Abstract Syntax Notation One (ASN.1) as defined in ITU X.680 and encoded with Octet Encoding Rules (OER) as defined in ITU X.696. By relying on ASN.1 we can take advantage of highly sophisticated tooling which allows us to verify the integrity of our specifications and the correctness of our implementations. By encoding with OER we ensure that parsers are very simple to write, wire formats are compact and encoding/decoding performance is excellent.

More information

- [ASN1 Introduction](#)
- [OER Overview](#).
- [Repository of Interledger ASN.1 modules](#)

PSK Data Format

The *PSK Data Format* is a data structure closely resembling [HTTP headers](#), defined for use with the PSK Interledger protocol. The same data format is used in the reference implementation of the IPR transport layer and in Mojaloop, because it provides some convenient properties. For example, it provides a public headers section so that all connectors in the middle of a payment can

ILP Packet

The ILP Packet is a binary data structure that should be attached to transfers (as a memo, if possible) to connect them to an Interledger payment.

- The [Interledger address](#) of the account where the payment should ultimately be delivered
- A 64-bit unsigned integer amount, with the scale and currency defined by the ledger where the amount is to be delivered
- An arbitrary, opaque, variable-length binary data field

More information

- [ILP Packet Definition](#)

ILP Error Format

The ILP Error Format is a binary data structure that Interledger components use to indicate a problem with executing a payment in the Interledger layer. The error format includes an error code, which is inspired by HTTP status codes, where the prefix specifies a broad category of causes and the number specifies the exact error that occurred. To distinguish Interledger error codes from HTTP status codes, Interledger errors use a letter prefix instead of a number. For example, temporary interledger errors use the prefix "T"-this is similar to HTTP status codes in the 500 range.

More information

- [ILP Error Format](#)

Amounts

In the Interledger layer, amounts are always represented as 64-bit unsigned integers. This provides extremely predictable precision and rounding behavior. Interledger amounts cannot be negative because you cannot transfer a negative value. (That would be the equivalent of a pull payment in a push payment system.) The amount is always defined in the context of a particular ledger, specifically, the one where the receiver's address is located. Each ledger's interface must define a translation from its internal data format to a 64-bit unsigned integer. In the [Five Bells Ledger](#)

[API](#), the **Get Metadata** method handles this by reporting the currency and scale of that currency.

Two ledgers may choose different scales for representing the same currency, depending on their intended use case. For example, a ledger optimized for micropayments might have a "nanodollar precision" with a minimum amount of 10^{-9} USD, while a traditional bank might set the limit at "millidollar precision" such that the minimum amount is 10^{-3} USD (\$0.001). In the nanodollar precision ledger, 2 USD would be represented as 2000000000 while in the millidollar precision ledger 2 USD would be 2000. Interledger's 64-bit unsigned integer can fit very large numbers without losing precision. For example, a payment in the amount of the gross national product of the USA in 2015 (18.14 trillion purchasing-parity dollars) could be represented down to the level of 10^{-6} dollars (\$0.000001) without rounding. In the unlikely event that a payment requires more precision than a 64-bit integer can provide, it could be divided into two Interledger payments to different ledger prefixes that represent different scales.

In JSON data, Interledger amounts should be represented as decimal strings. Many JSON parsers assume JSON numbers have the same precision as JavaScript numbers (64-bit double-precision floating point), which cannot represent all unsigned 64-bit integers without losing precision. By representing amounts as strings, senders and receivers of JSON can serialize and deserialize the amounts using data types that can represent the full precision, or at least as much precision as is necessary for their specific purposes.

Connector Risks

Interledger guarantees that the receiver gets paid or the sender gets their money back. There are, however, some failure cases where connectors in the middle could lose money because a transfer expired before the connector could execute it. (In those cases, the receiver gets paid *and* the sender gets their money back, thanks to the backwards execution.) To balance out these risks, connectors should plan accordingly.

[IL-RFC-18: Connector Risk Mitigations](#) discusses general patterns connectors can follow to minimize their risk of losing money.

In the case of a Mojaloop instance, some more specific modifications are possible to further mitigate risk. These are possible because Mojaloop design involves a trusted central ledger, and each DFSP has control over its ledger *and* connector. These optimizations are, in short:

- *Receiver Wait & Pay* - The receiving DFSP tries to fulfill the transfer on the central ledger before preparing the transfer to the final receiver.
- *Sender Check-before-Rollback* - The sending DFSP checks the outcome of the transfer on the central ledger shortly after that transfer expires. The sending DFSP sets the timeout of the transfer in its own ledger such that it can execute the transfer (including possible retries) after seeing the outcome on the central ledger.
 - Or the sender checks the result on the central ledger before expiring a transfer in the

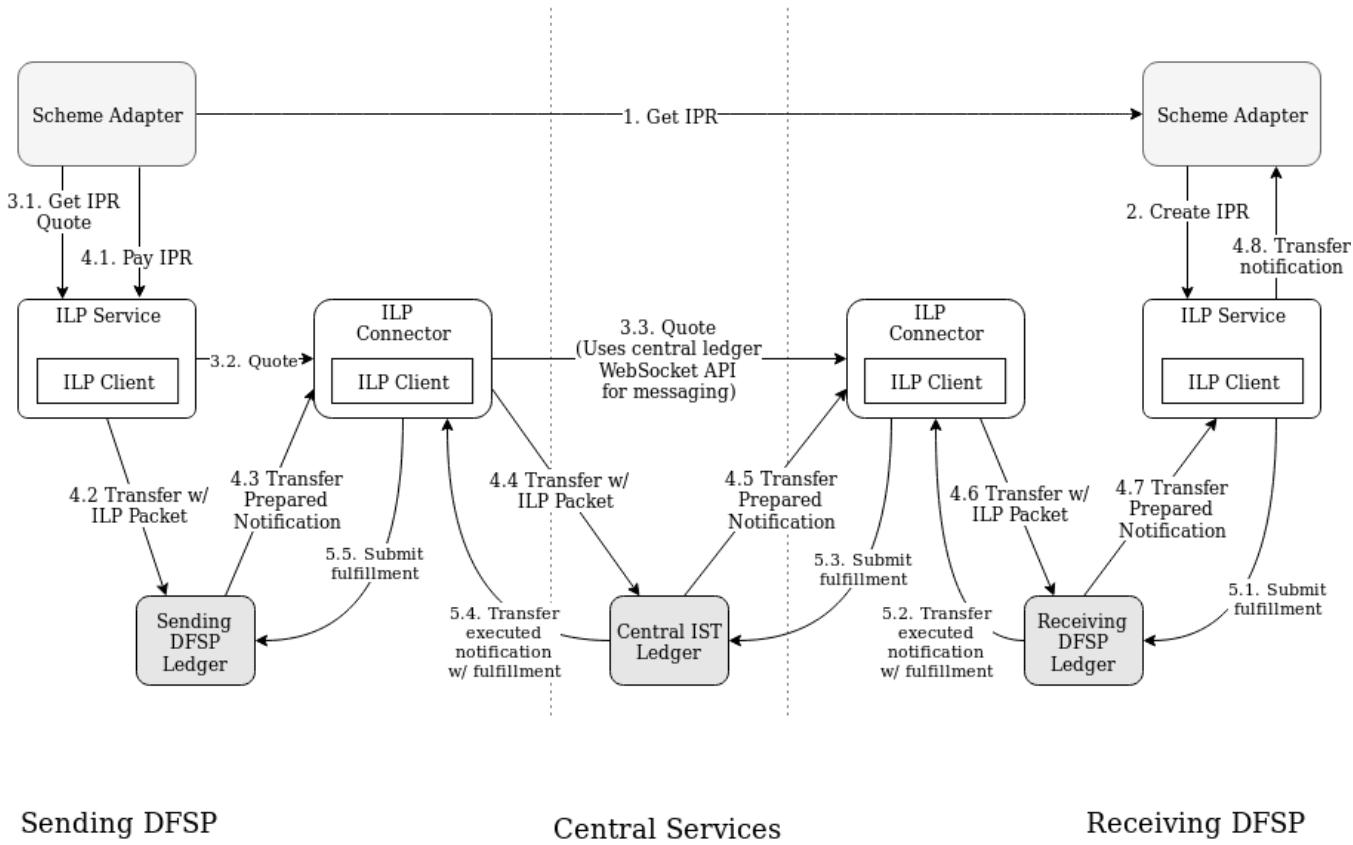
sending DFSP's ledger. If the transfer succeeded on the central ledger, the sender executes the transfer on the sender's DFSP ledger even if that means the transfer in the sending DFSP's ledger executes at or slightly past its expiration time.

How to Troubleshoot ILP Payment Issues

- Consult the logs.
- See which transfers executed and which transfers failed.

Software Components

Components appear and are described in the following diagram and table.



Sending DFSP

Central Services

Receiving DFSP

Component Name	Summary
<u>Scheme Adapter</u>	A custom application defined by Mojaloop that handles the planning of payments on an end-to-end basis.
<u>ILP Service</u>	A RESTful service that acts as a client library for accessing ILP-related functions, such as getting quotes from the ILP Connector, generating conditions in IPR format, and similar tasks.
<u>ILP Client</u>	A common library used within the ILP Service and the ILP Connector.
<u>ILP Connector</u>	Holds money in the DFSP ledger and the Central Ledger, and sets the rates of exchange between them.
<u>DFSP Ledger</u>	The reference ledger for a DFSP. Tracks customer balances and exposes the Five Bells Ledger API .
<u>Central Ledger</u>	The reference ledger for a central IST. Tracks DFSP / connector balances and exposes the Five Bells Ledger API .

Scheme Adapter

A custom application defined by Mojaloop that handles the planning of payments on an end-to-end basis. Part of the DFSP software.

More information

- [Scheme Adapter Repository](#)

ILP Service

A convenience application for Mojaloop that provides a handful of ILP-related functions through a RESTful(-ish) API. Functionality includes getting quotes, creating Interledger Payment Request objects (which include the cryptographic conditions), and issuing notifications when the connector detects that ILP-compatible transfers have been prepared.

More information

- [ILP Service Repository](#)

ILP Client

This is a standard library used by the ILP Service and the ILP Connector. It handles things like validating and verifying [Crypto-Conditions][]. It interfaces with the ILP Connector using the Interledger Protocol and the Interledger Quoting Protocol. It also interfaces with the reference DFSP ledgers and central IST ledger using the [Five Bells Ledger API](#), and can be extended with plugins for other ledger interfaces.

More information

- [ILP Client Repository](#)
- [Five Bells Ledger Plugin Repository](#)

ILP Connector

The ILP Connector connects one DFSP's Ledger to another ledger. For now, a DFSP's connector always connects the DFSP to the Central Ledger. In the future, it is possible that ILP Connectors could connect two DFSP ledgers directly, and there could even be a competitive marketplace of ILP Connectors between pairs of ledgers.

The ILP Connector has accounts holding money with the ILP Ledger Adapters of each of the two ledgers it connects. (It can also connect directly to ledgers that implement ILP natively.) The connector defines the exchange rates between balances on the two ledgers.

Mojaloop uses the Interledger project's reference implementation for a connector.

More information

- [ILP Connector Repository](#)

Running the ILP Software

[Ansible](#) is used for deploying the [ilp-connector](#) and the [ilp-service](#). The [Ansible Playbook](#) can be run with the command:

```
ansible-playbook -v --extra-vars="docker_username=<FILL ME IN>
docker_password=<FILL ME IN> docker_email=<FILL ME IN>" --inventory-
file=hosts-test ansible.yml
```

This command should be run from the [ansible](#) directory in this repository.

Ansible uses the SSH keys found in your normal SSH directory to log in to the servers.

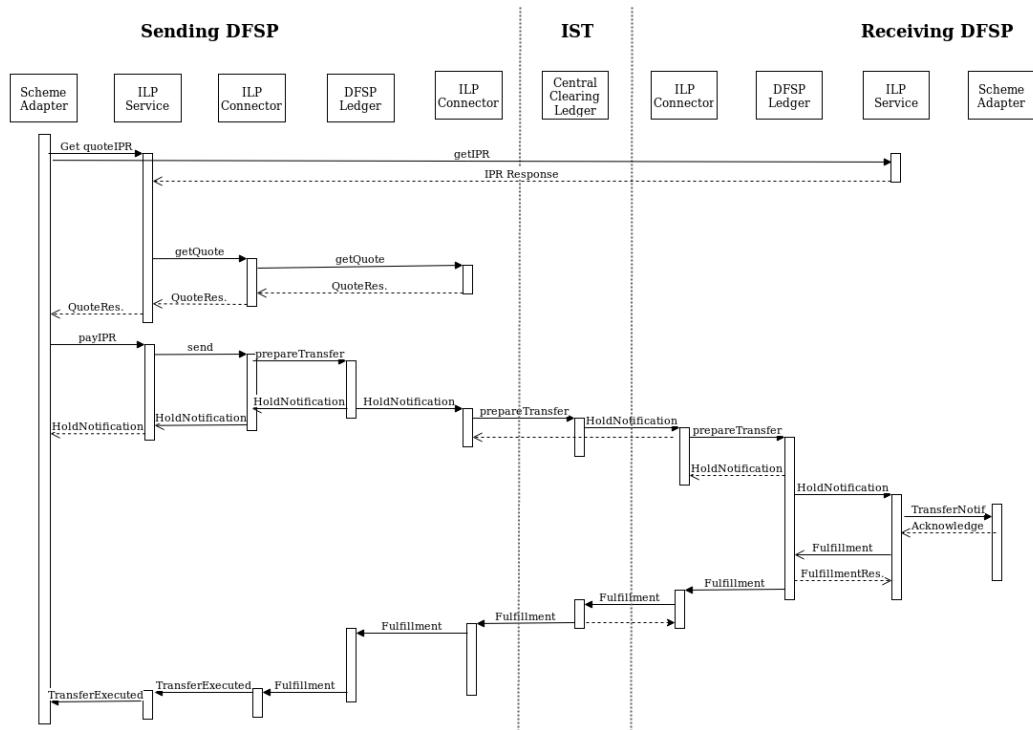
The Docker credentials are those used for the private registry (modusbox-level1-docker.jfrog.io).

The Inventory File should either be the [hosts-test](#) or [hosts-qa](#) depending on whether you want to deploy the components to the L1P Test or QA environment.

Execution Flow

Execution flow is shown in the following diagram.

L1P Interledger Flow



ilp-service

An Interledger Protocol (ILP) sending and receiving client with a RESTful(ish) API, designed for the Mojaloop.io project.

The `ilp-service` is designed to be used to build Interledger-capable systems on top of the [Interledger Payment Request \(IPR\)](#) transport protocol. For more details on the Interledger protocol suite layers, see [IL-RFC 1: Interledger Architecture](#).

Contents:

- [Deployment](#)
- [Configuration](#)
- [API](#)
- [Logging](#)
- [Tests](#)
- [Notes for Application Layer Protocol Implementors](#)

Deployment

You can deploy the `ilp-service` with the [ILP Ansible Playbook](#).

Configuration

The `ilp-service` is configured using environment variables. For deployment, [ILP Ansible Playbook](#) sets the values of these variables.

Environment Variable	Type	Description
<code>ILP_SERVICE_PORT</code>	Number	Port to run the ILP Service on.
<code>ILP_SERVICE_ILP_PREFIX</code>	ILP	ILP address prefix for the DFSP. This should

Environment Variable	Type	Description
	Address Prefix	start with <code>private.</code>
<code>ILP_SERVICE_LEDGER_ADMIN_ACCOUNT</code>	<u>URI</u>	Admin ledger account URI. Used to enable the <code>ilp-service</code> to send quote requests and transfers on behalf of DFSP account holders.
<code>ILP_SERVICE_LEDGER_ADMIN_USERNAME</code>	String	Admin username. Used to enable the <code>ilp-service</code> to send quote requests and transfers on behalf of DFSP account holders.
<code>ILP_SERVICE_LEDGER_ADMIN_PASSWORD</code>	String	Admin password. Used to enable the <code>ilp-service</code> to send quote requests and transfers on behalf of DFSP account holders.
<code>ILP_SERVICE_CONNECTOR_ACCOUNT</code>	<u>URI</u>	Ledger account URI of the connector used to send Interledger payments.
<code>ILP_SERVICE_BACKEND</code>	<u>URI</u>	Backend server to send notifications to. See Backend Notifications .
<code>ILP_SERVICE_SKIP_BACKEND REVIEW</code>	Boolean	(Optional, mainly for debugging purposes) Skip the backend review and fulfill all incoming payments
<code>ILP_SERVICE_SECRET</code>	String	(Optional) Secret value used to create and verify IPRs as the receiver. If omitted, generates a random value on startup.
<code>ILP_SERVICE_CENTRAL_CONNECTOR_ACCOUNT</code>	<u>URI</u>	Account URI of the connector on the ledger one hop before the ledger that the admin is on (the IST or Central Ledger).
<code>ILP_SERVICE_CENTRAL_CONNECTOR_PASSWORD</code>	String	Account password of the connector on the ledger one hop before the ledger that the admin is on (the IST or Central Ledger).

Ledger URIs

A "Ledger account URI" is the HTTPS URL of an account in the [ILP Ledger Adapter API](#).

API

The sending side uses the following methods:

- [GET /quoteSourceAmount](#)
- [GET /quoteIPR](#)
- [POST /payIPR](#)

The receiving side uses the following methods:

- [POST /createIPR](#)
- [GET /ilpAddress](#)
- [Backend Notifications](#)

GET /quoteSourceAmount

Get a fixed source amount quote.

quoteSourceAmount Request

GET /quoteSourceAmount

The request must include the following query string parameters:

Parameter	Type	Description
destinationAddress	ILP Address	ILP Address for the destination. Note: This must be communicated in the Application Layer protocol to enable the sender to request quotes with fixed source amounts.
sourceAmount	Decimal String	Amount the source account would send to the connector, denominated in the currency of the source ledger.
destinationScale	Integer	Scale of the amounts on the destination ledger. Used to format the destinationAmount in the return value. This SHOULD be reported by the ledger.

Parameter	Type	Description
		Note: This must be communicated in the Application Layer protocol. If this value is incorrect, the result will be off by several orders of magnitude.
connectorAccount	URI	(Optional) Ledger account URI of a connector to send the quote request to. If omitted, <code>ilp-service</code> uses the connector(s) defined in its config.

quoteSourceAmount Response

A successful response uses the HTTP status code **200 OK**. The message body is a JSON object with the following fields:

Parameter	Type	Description
destinationAmount	Decimal String	Amount the destinationAddress will receive, denominated in the currency of the destination ledger.
connectorAccount	URI	Ledger account URI of the connector through which this payment can be sent.
sourceExpiryDuration	Integer	Number of seconds after the payment is submitted that the outgoing transfer will expire.

GET /quoteIPR

Get a quote for an [Interledger Payment Request \(IPR\)](#) communicated from a receiver.

quoteIPR Request

GET /quoteIPR

The request must include the following query string parameters:

Parameter	Type	Description
ipr	Base64-URL String	Binary IPR from the receiver.
connectorAccount	URI	(Optional) Ledger account URI of a connector to send the quote request to. If omitted, <code>ilp-service</code> uses the connector(s) defined in its config.

quoteIPR Response

A successful response uses the HTTP status code **200 OK**. The message body is a JSON object with the following fields:

Parameter	Type	Description
sourceAmount	Decimal String	Amount the source amount should send to the connector, denominated in the currency of the source ledger.
connectorAccount	URI	Ledger account URI of the connector through which this payment can be sent.
sourceExpiryDuration	Integer	Number of seconds after the payment is submitted that the outgoing transfer expires.

POST /payIPR

Execute an Interledger payment through a connector.

This method is idempotent. Submitting the same IPR multiple times does not cause the transfer to be executed multiple times. However, you cannot use this method to retry paying for the same IPR.

Note: This method prepares the transfer in the ledger using the unique `paymentID` embedded in the binary `ipr` of this request as the ID of the transfer. The ledger MUST ensure that the specified transfer ID is unique, or the guarantee that this transfer is idempotent does not hold.

payIPR Request

POST /payIPR

The request's message body is a JSON object with the following fields:

Parameter	Type	Description
ipr	Base64-URL String	Binary IPR from the receiver.
sourceAmount	Decimal String	Amount the source account should send to the connector, denominated in the currency of the source ledger.
sourceAccount	URI	Ledger account URI to send the transfer from.
connectorAccount	URI	Ledger account URI of the connector through which this payment

Parameter	Type	Description
		will be sent.
sourceExpiryDuration	Integer	Number of seconds after the payment is submitted that the outgoing transfer will expire.

payIPR Response

A successful response uses the HTTP status code **200 OK**. The message body is a JSON object with the following fields:

Parameter	Type	Description
paymentId	UUID	The UUID of the local transfer and the Interledger payment (generated by the receiver and included in the IPR).
connectorAccount	URI	Ledger account URI of the connector through which this payment was sent.
status	String	executed, rejected, expired
rejectionMessage	JSON Object	Only present when status is rejected. This SHOULD be an object in the ILP Error Format .
fulfillment	Base64-URL String	Only present when status is executed

POST /createIPR

Create an [Interledger Payment Request \(IPR\)](#). The IPR should be communicated from a receiver to a sender using an Application Layer protocol to request a payment from the sender.

Once an IPR has been created, the `ilp-service` will listen for incoming prepared transfers. The `ilp-service` sends notifications to the backend to determine if an incoming payment should be accepted.

createIPR Request

```
POST /createIPR
```

The request includes the following query string parameters:

Parameter	Type	Description
paymentId	UUID	Unique ID of this payment request. This will be included in the notifications about incoming transfers and should be used to correlate IPRs communicated to senders with incoming payments.
destinationAccount	<u>URI</u>	Ledger account URI of the account into which the funds will be paid.
destinationAmount	Decimal String	The requested amount, which will be paid into the destinationAccount.
expiresAt	ISO Timestamp	Expiration of the payment request. Incoming transfers received after this time will be automatically rejected.
data	Object	(Optional) Arbitrary JSON data to attach to the IPR.

TODO: Would you rather pass in the absolute ISO timestamp or the relative time in seconds?

createIPR Response

A successful response uses the HTTP status code **200 OK**. The message body is a JSON object with the following fields:

Parameter	Type	Description
ipr	Base64-URL String	Binary IPR that should be communicated to the sender over an Application Layer protocol.

GET /ilpAddress

Get the ILP Address for a given ledger account. **Note: This should be included in Application Layer protocols so that senders can request fixed source amount quotes to the receiver.**

ilpAddress Request

```
GET /ilpAddress
```

The request includes the following query-string parameters:

Parameter	Type	Description
account	<u>URI</u>	Ledger account URI.

ilpAddress Response

A successful response uses the HTTP status code **200 OK**. The message body is a JSON object with the following fields:

Parameter	Type	Description
ilpAddress	ILP Address String	The ILP Address corresponding to the account provided. This is used primarily for quoting by source amount.

Backend Notifications

When a transfer is prepared, the `ilp-service` send a notification callback in the form of outgoing HTTP requests. A client of the `ilp-service` must run a service that accepts HTTP requests in the notification format and processes them accordingly. Most importantly, the response to these notification requests MUST use the HTTP status code **200 OK** to indicate success. If the response uses any other status code, the `ilp-service` rejects the transfer using the response's message body as the rejection reason.

Notification Request

When a transfer is prepared, the `ilp-service` sends a notification to the configured backend service at the URL specified in the `ILP_SERVICE_BACKEND` configuration option.

```
POST <ILP_SERVICE_BACKEND>/notifications
```

Notification Request Body

The message body of the Notification request is a JSON Object with the following fields:

Parameter	Type	Description
ipr	Base64-URL String	The IPR being paid by this transfer.
paymentId	UUID String	Unique ID of the payment request. This is the same value that was submitted when the IPR was created.
destinationAccount	URI	Ledger account URI of the account into which the funds will be paid if the transfer is executed.

Parameter	Type	Description
status	String. One of: prepared, executed	The status of the payment. See below for more details.
fulfillment	Base64-URL String	This is the cryptographic fulfillment used to execute the incoming transfer. (Only present when status is <code>executed</code> .)
data	Object	Arbitrary JSON data attached to the IPR. (Only present when <code>data</code> was passed into the corresponding <code>createIPR</code> call)

Transfer Status Descriptions

The `status` field of the payment is one of the following values:

Status	Description
<code>prepared</code>	Funds are on hold and the <code>ilp-service</code> is awaiting approval from the backend before fulfilling the transfer condition. The transfer will only be fulfilled if the backend responds to this notification with the HTTP status code 200 OK .
<code>executed</code>	The incoming transfer has been executed and the <code>destinationAccount</code> has been paid. The transfer has a <code>fulfillment</code> field, which contains the cryptographic fulfillment used to execute the incoming transfer.

Notification Response

If the notification is processed successfully, the response MUST use the HTTP Status Code **200 OK**. The `ilp-service` ignores the headers and message body of this response.

If there is a problem, the response SHOULD have the following attributes:

Attribute	Value
HTTP status code	Any valid HTTP Status Code in the range 400-599 (inclusive)
Content-Type	<code>application/json</code>
Header	
Message body	JSON object describing the nature of the problem. The <code>ilp-service</code> uses this value as the rejection reason of the transfer. This SHOULD be an object in the ILP Error Format .

Logging

TODO: What gets logged and where (forensic & generic logs)

Tests

Running the tests:

```
npm run test
```

Tests include code coverage via [istanbul](#) and unit tests via [mocha](#). See the [test/](#) folder for testing scripts.

Notes for Application Layer Protocol Implementors

The `ilp-service` is designed to be used to build Interledger payments and the Interledger Payment Request (IPR) Transport Layer protocol into Application Layer protocols. This section includes some notes and considerations for implementors of Application Layer protocols using this service.

ILP Addresses

The Interledger Protocol uses [ILP Addresses](#) to identify ledgers and accounts in a scheme-agnostic manner.

Most of the use of ILP Addresses is handled by the `ilp-service`, with one exception. Application Layer protocol implementors SHOULD include a way for a sender to get the ILP address of a receiver to enable [quoting with a fixed source amount](#). The receiver can use the `ilp-service` to [get the ILP address for a given ledger account](#).

Interledger Payment Requests (IPRs)

The Interledger Payment Request (IPR) includes a fixed destination amount, the destination ILP address, and payment

condition. Implementors of Application Layer protocols MUST include a way for the receiver to communicate the IPR to the sender.

Complex Fees

The IPR includes a fixed destination amount. The sender will get a quote from one or more connectors to determine the cost of delivering the specified amount to the receiver.

If receiving DFSPs want to implement additional fees, those can be included in the Application Layer protocol communication and deducted from the end recipient's account after transfers are executed.

If sending DFSPs want to implement additional fees, those can be added to the chosen source amount or the source amount determined by quoting an IPR. The fee amount can be deducted from the sender's account when the outgoing transfer is executed.

Interop Services

The various interop service APIs act as proxies and/or provide features such as validation, authentication and data transformation where necessary. The services operate based on service specifications provided in both Open API and RAML. The services run on Mule community runtime. There are three interop micro-services that are mentioned below and several supporting projects.

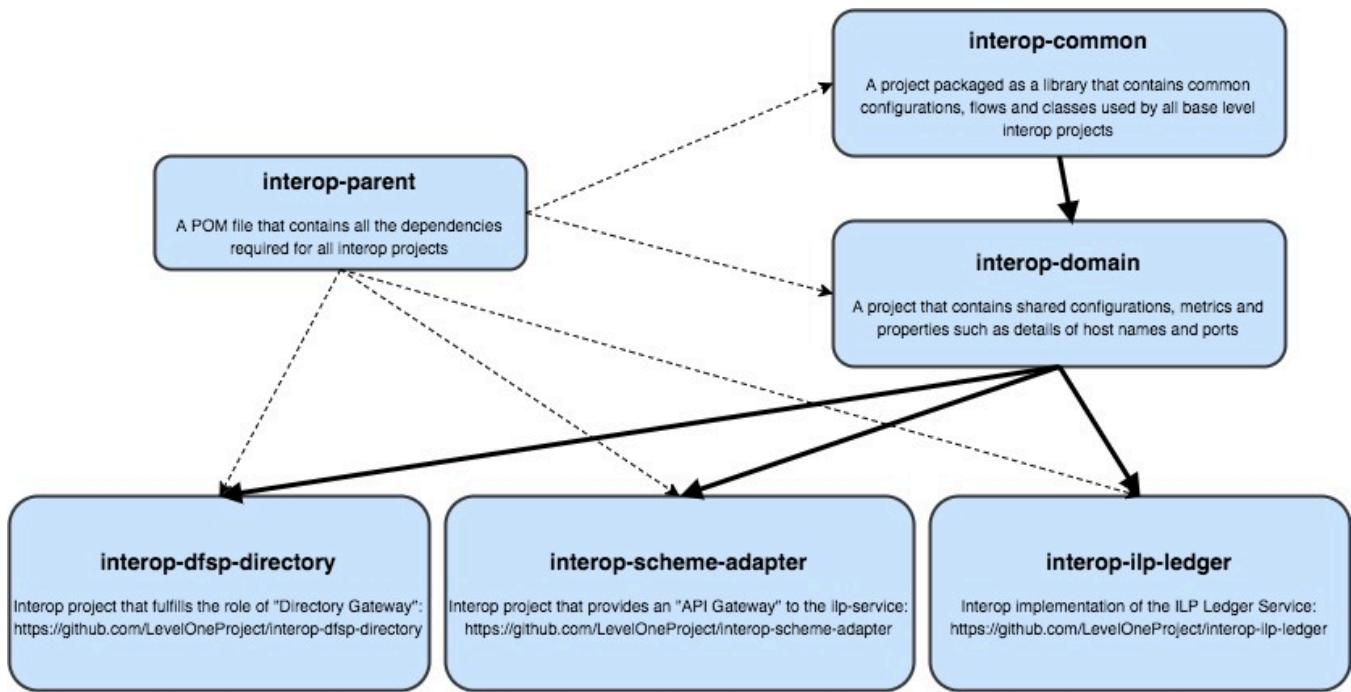
There are several repositories that support non-functional requirements such as Performance, Logging, Metrics and Deployment that are discussed below as well.

Contents:

- [Overview](#)
- [Architecture](#)
- [Interfaces](#)
- [Testing](#)
- [Security](#)
- [Resilience](#)
- [Performance](#)
- [Logging](#)
- [Deployment](#)

Overview

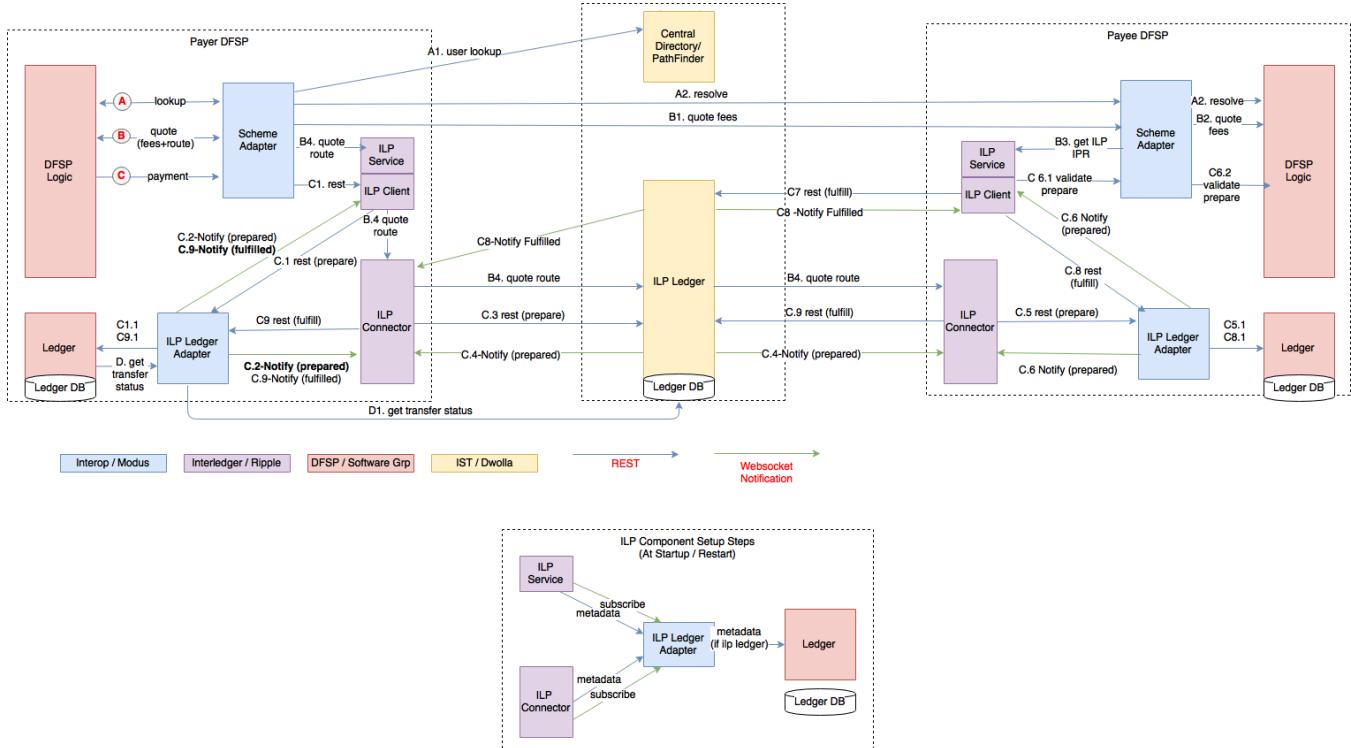
Structure of interop services



Architecture

User Message/Flow Diagram of L1P System

The diagram in this section (L1P Reference Implementation) show the positive or "happy" path of the user. Negative and boundary cases are described in other specifications. A data flow diagram is also used for threat modeling.



Interfaces

- **interop-dfsp-directory** - This project provides an API gateway to the IST Directory Naming Service and provides resources to - "get metadata about a directory", "get customer information by providing identifier, identifierType", "Register a DFSP" and "get identifierTypes supported by the central directory"
 - [Spec](#) | [Code](#)
- **interop-ilp-ledger** - This project provides an interop API implementation of ILP Ledger Service.
 - [Spec](#) | [Code](#)
- **interop-scheme-adapter** - This project provides an API gateway to the ilp-service microservice. It supports methods to query, quote, payment request as specified in the [ilp-service](#).
 - [Spec](#) | [Code](#)

Supporting projects for functionality

- [interop-parent](#)
- [interop-common](#)
- [interop-domain](#)

Projects for non-functional requirements

- [interop-devops](#)
- [interop-elk](#)

Testing

Java Unit Tests exist for each of the projects for unit testing, some of which use WireMock framework. Tests are run as part of executing the Maven pom.xml as mvn clean package.

Security

Security/Threat Model for L1P Reference Implementation is [here](#)

Resilience

Resilience model for L1P Reference Implementation is [here](#)

Performance

Performance approach for Interop services and for L1P project as a whole is described [here](#). Tools used and ways to perform analysis are also described. This can be used for use case or scenario tests as well as isolated testing of services using mocks. The same approach and scripts can be used for Load Testing as well.

Logging

Logging guidelines for L1P project were drafted and after review by partner teams, documented [here](#). Aspects of end-to-end Tracing and support for Metrics are discussed and requirements described in the guidelines document. The configuration used and other customizations such as adding indexes can be found in the [interop-elk](#) project.

Deployment

The L1P system can be deployed using Vagrant and Ansible playbooks to create two DFSPs and one CST VMs with support for MGMT VMs to allow execution on all supported platforms, including windows. The user guide for this is [here](#)

Interop Ledger

This project provides an interop API implementation of ILP Ledger Service.

Contents:

- [Deployment](#)
- [Configuration](#)
- [API](#)
- [Logging](#)
- [Tests](#)

Deployment

Project is built using Maven and uses Circle for Continous Integration.

Installation and Setup

Anypoint Studio

- <https://www.mulesoft.com/platform/studio>
- Clone <https://github.com/mojaloop/interop-ilp-ledger.git> to local Git repository
- Import into Studio as a Maven-based Mule Project with pom.xml
- Go to Run -> Run As Configurations. Make sure interop-ilp-ledger project is highlighted.

Standalone Mule ESB

- <https://developer.mulesoft.com/download-mule-esb-runtime>
- Add the environment variable you are testing in (dev, prod, qa, etc). Open /conf/wrapper.conf and find the GC Settings section. Here there will be a series of wrapper.java.additional.(n) properties. create a new one after the last one where n=x (typically 14) and assign it the next number (i.e., wrapper.java.additional.15) and assign -DMULE_ENV=dev as its value (wrapper.java.additional.15=-DMULE_ENV=dev)
- Download the zipped project from Git
- Copy zipped file (Mule Archived Project) to /apps

Run Application

Anypoint Studio

- Run As Mule Application with Maven

Standalone Mule ESB

- CD to /bin -> in terminal type ./mule

Configuration

[pom.xml](#) and [circle.yml](#) can be found in the repo, also linked here

API

The API provides methods for the following: *Find an account by ID* Add an account by ID *List all accounts* Find a local transfer by id *Request a transfer* Execute a transfer * Retrieve all accounts of all connectors on a ledger

Below are the APIs for reference *RAML* [here](#) *OpenAPI* [here](#)

Logging

Server path to logs is: /logs/interop-ilp-ledger.log for example: /opt/mule/mule-dfsp1/logs/interop-ilp-ledger.log

Currently the logs are operational and include information such as TraceID and other details related to the calls or transactions such as path, method used, header information and payer/payee details.

Tests

Java Unit Test exist for the project and include test for:

- Invalid path should return 404
- Put accounts
- Get account
- Get transfer
- Get health
- Reject transfer should get return valid response
- Get metadata
- Put transfer fulfillment
- Get connectors
- Post messages

Anypoint Studio

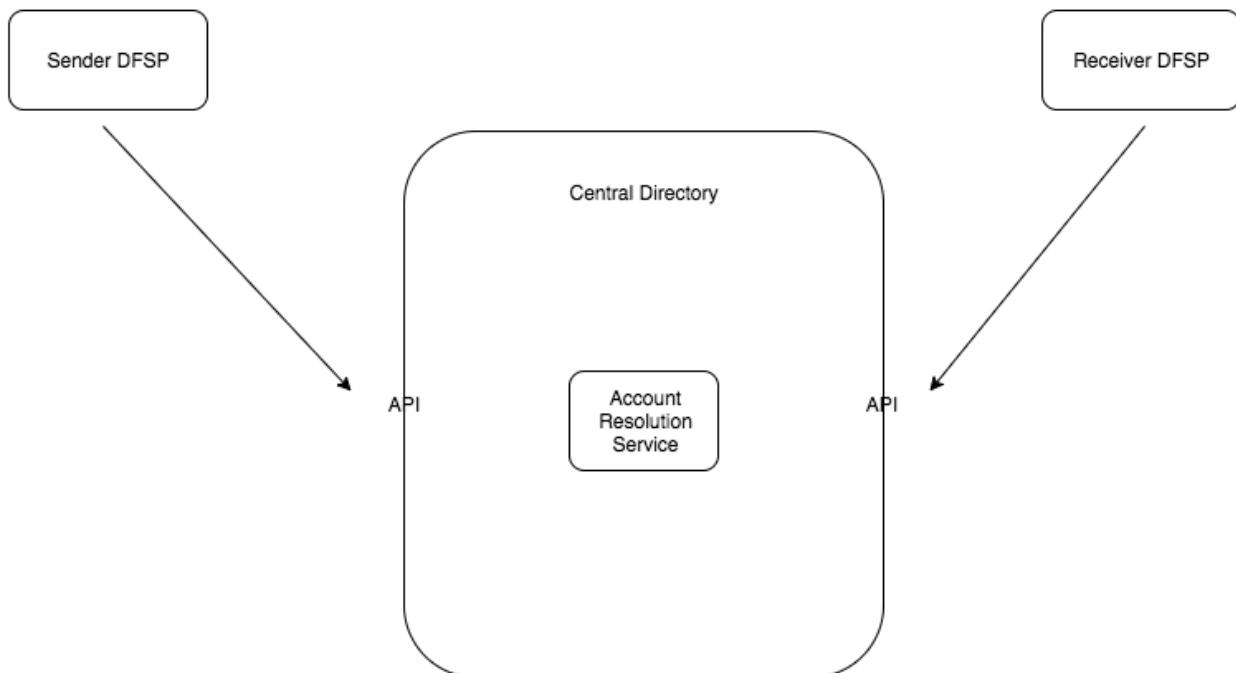
- Run Unit Tests
- Test API with Anypoint Studio in APIKit Console
- Verify Responses in Studio Console output

Tests are run as part of executing the Maven pom.xml as mvn clean package. Also, test can be run by running com.llp.interop.spsp.ILPLedgerAdapterFunctionalTest java class as a Test.

Central Directory

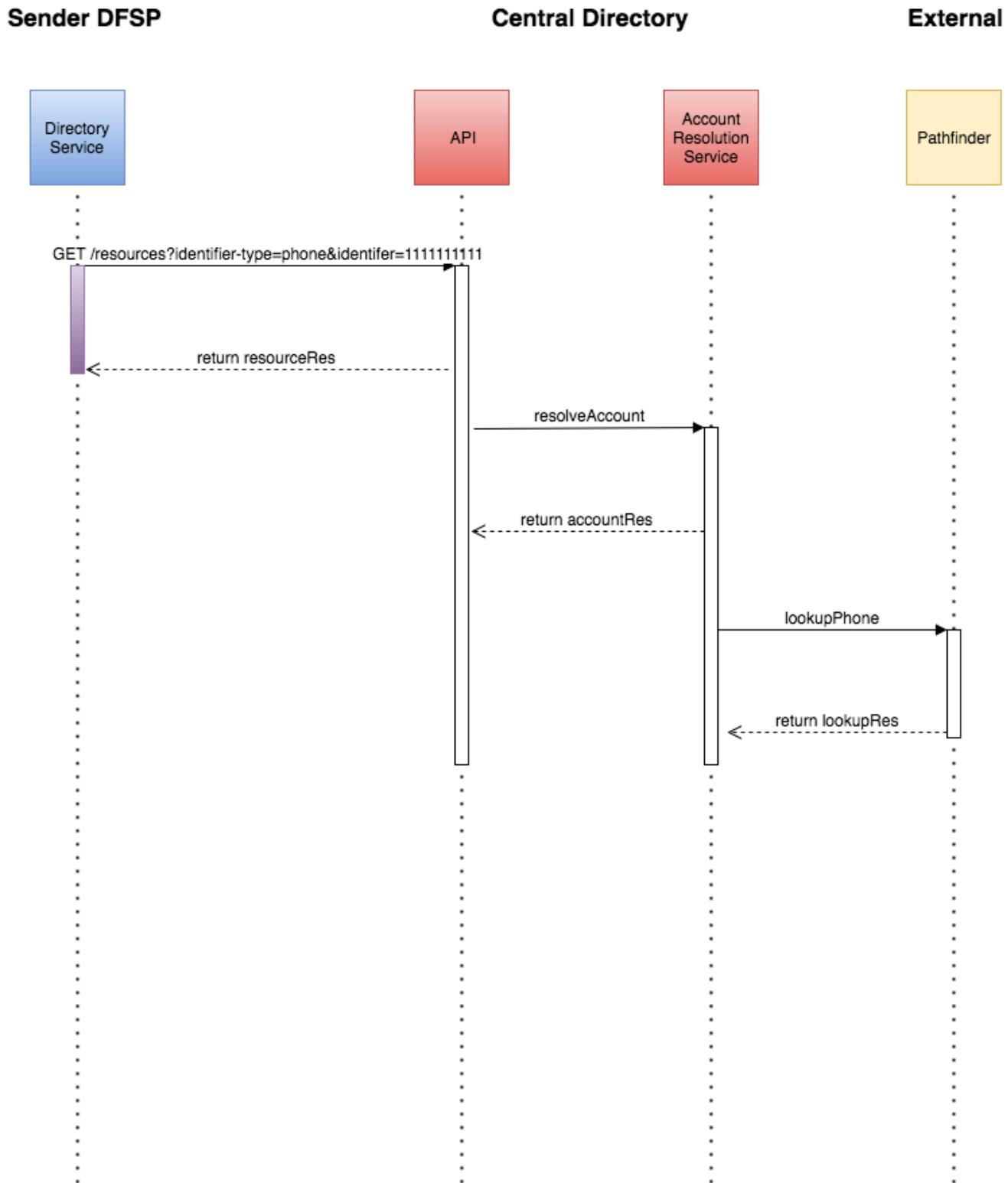
Central Directory

Component Architecture



Component	Function	Interaction
API	Handles calls for account resolution and name retrieval.	Calls internal services to resolve an End User's account.
Account Resolution Service	Handles storing the User Number and any associated Account information (e.g. ILP addresses), and retrieving the Account by User Number.	Called by API; called by User Identification Service when looking up End User by Account.
User Identification Service	Handles retrieving the End User name by User Number, Account, etc.	Called by API

End User Lookup



Directory Endpoints

In this guide, we'll walk through the different central directory endpoints: *POST [Register a DFSP](#)* *GET [Get identifier types](#)* *POST [Register an identifier](#)* *GET [Lookup resource by identifier](#)* *GET [Get directory metadata](#)* *GET [Directory health check](#)*

The different endpoints often deal with these data structures: *[Resource Object](#)* *[DFSP Object](#)* *[Identifier Type Object](#)* *[Metadata Object](#)*

Information about various errors returned can be found here: * [Error Information](#)

Endpoints

Register a DFSP

This endpoint allows a DFSP to be registered to use the central directory.

HTTP Request

```
POST http://central-directory/commands/register
```

Authentication

Type	Description
HTTP Basic	The username and password are admin:admin

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request body

Field	Type	Description
name	String	The name of the created DFSP
shortName	String	The shortName of the created DFSP
providerUrl	String	The url reference for the DFSP

Response 201 Created

Field	Type	Description
Object	DFSP	The <u>DFSP object</u> as saved

Request

```
POST http://central-directory/commands/register HTTP/1.1
Content-Type: application/json
{
  "name": "The first DFSP",
  "shortName": "dfsp1",
  "providerUrl": "http://url.com"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "name": "The first DFSP",
  "shortName": "dfsp1",
  "providerUrl": "http://url.com",
```

```
"key": "dfsp_key",
"secret": "dfsp_secret"
}
```

Errors (4xx)

Field	Description
AlreadyExistsError	The DFSP already exists (determined by name)

```
{
  "id": "AlreadyExistsError",
  "message": "The DFSP already exists (determined by name)"
}
```

Get identifier types

This endpoint allows retrieval of the identifier types supported by the central directory.

HTTP Request

```
GET http://central-directory/identifier-types
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1

Response 200 OK

Field	Type	Description
Object	Array	List of supported <u>Identifier Type objects</u>

Request

```
GET http://central-directory/identifier-types HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "identifierType": "eur",
    "description": "Central end user registry"
  }
]
```

Register an identifier

This endpoint allows a DFSP to add an identifier associated with their account. When the identifier is retrieved from the [Lookup resource by identifier](#) endpoint, the url registered with the DFSP will be returned.

HTTP Request

```
POST http://central-directory/resources
```

Authentication

Type	Description
HTTP Basic	The key and secret for the DFSP

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request body

Field	Type	Description
identifier	String	The identifier type and identifier to be created, separated by a colon
preferred	String	(optional) Sets the identifier as preferred, can either be <i>true</i> or <i>false</i> .

Preferred will default to true if it is the first DFSP added for this identifier, and will default to false if another DFSP already has been added.

If the current DFSP being updated is preferred and the preferred value is set to false, an error will be thrown.

Response 201 Created

Field	Type	Description
Object	Resource	The newly-created <u>Resource object</u> as saved

Request

```
POST http://central-directory/resources HTTP/1.1
Content-Type: application/json
{
  "identifier": "eur:dfsp123",
  "preferred": "true"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "name": "The First DFSP",
  "providerUrl": "http://dfsp/users/1",
  "shortName": "dsfp1",
  "preferred": "true",
  "registered": "true"
}
```

Errors (4xx)

Field	Description
AlreadyExistsError	The identifier has already been registered by this DFSP

```
{
  "id": "AlreadyExistsError",
  "message": "The identifier has already been registered by this
```

```
DFSP"  
}
```

Lookup resource by identifier

This endpoint allows retrieval of a URI that will return customer information by supplying an identifier and identifier type.

HTTP Request

```
GET http://central-directory/resources?identifier={identifierType:identifier}
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1

Query Params

Field	Type	Description
identifier	String	Valid identifier type and identifier separated with a colon

Response 200 OK

Field	Type	Description
Object	Array	An array of <u>Resource objects</u> retrieved

The returned array will contain one DFSP with preferred set to true. All others should be set to false.

Request

```
GET http://central-directory/resources?identifier=eur:1234 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "name": "The First DFSP",
    "providerUrl": "http://dfsp/users/1",
    "shortName": "dsfp1",
    "preferred": "true",
    "registered": "true"
  },
  {
    "name": "The Second DFSP",
    "providerUrl": "http://dfsp/users/2",
    "shortName": "dsfp2",
    "preferred": "false",
    "registered": "false"
  }
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found.

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Get directory metadata

Returns metadata associated with the directory

HTTP Request

```
GET http://central-directory
```

Response 200 OK

Field	Type	Description
Metadata	Object	The Metadata object for the directory

Request

```
GET http://central-directory HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "directory": "http://central-directory",
  "urls": {
    "health": "http://central-directory/health",
    "identifier_types": "http://central-directory/identifier-types",
    "resources": "http://central-directory/resources",
    "register_identifier": "http://central-directory/resources"
  }
}
```

Directory Health

Get the current status of the service

HTTP Request

```
GET http://central-directory/health
```

Response 200 OK

Field	Type	Description
status	String	The status of the ledger, <i>OK</i> if the service is working

Request

```
GET http://central-directory/health HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "status": "OK"
}
```

Data Structures

Resource Object

A resource represents the information returned about an identifier and identifier type.

A resource object can have the following fields:

Name	Type	Description
name	String	Name of the DFSP
providerUrl	URI	A URI that can be called to get more information about the customer
shortName	String	Shortened name for the DFSP
preferred	String	Details if the DFSP is set as preferred, can either be <i>true</i> or <i>false</i>
registered	String	Returns <i>true</i> if DFSP is registered for the identifier, <i>false</i> if defaulted

DFSP Object

Represents a DFSP that has registered with the central directory.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. A DFSP object can have the following fields:

Name	Type	Description
name	String	The name of the created DFSP
shortName	String	The shortName of the created DFSP
providerUrl	String	The URL for the DFSP
key	String	Key used to authenticate with protected endpoints. Becomes the username for Basic Auth. Currently the same value as the name field
secret	String	Secret used to authenticate with protected endpoints. Currently the same value as the name field

Identifier Type Object

Represents an identifier type that is supported by the central directory.

An identifier type object can have the following fields:

Name	Type	Description
identifierType	String	Unique name of the identifier type
description	String	Description of the identifier type

Metadata Object

The central directory will return a metadata object about itself allowing client's to configure themselves properly.

A metadata object can have the following fields:

Name	Type	Description
directory	URI	The directory that generated the metadata
urls	Object	Paths to other methods exposed by this directory. Each field name is short name for a method and the value is the path to that method.

Error information

This section identifies the potential errors returned and the structure of the response.

An error object can have the following fields:

Name	Type	Description
id	String	An identifier for the type of error
message	String	A message describing the error that occurred
validationErrors	Array	<i>Optional</i> An array of validation errors
validationErrors[].message	String	A message describing the validation error
validationErrors[].params	Object	An object containing the field that caused the validation error
validationErrors[].params.key	String	The name of the field that caused the validation error
validationErrors[].params.value	String	The value that caused the validation error
validationErrors[].params.child	String	The name of the child field

```
HTTP/1.1 404 Not Found
Content-Type: application/json
{
  "id": "InvalidQueryParameterError",
  "message": "Error validating one or more query parameters",
  "validationErrors": [
    {
      "message": "'0' is not a registered identifierType",
      "params": {
        "key": "identifierType",
        "value": "0"
      }
    }
  ]
}
```

User Discovery

In order for a sender to make a payment to a receiver they must "discover" some things about the receiver, such as their ILP address and the currency of their account.

Following this, further information may be required by the sending system such as the full name, public key or even a photograph of the receiver for validation by the sender.

Flow

1. The sender (End User) has an identifier for the receiver such as an account number, mobile number, or national identity number.
2. The sending system (DFSP) uses this identifier to discover the information required to:
 - a. Show appropriate info on the phone of the sender (name of receiver, currency code of receiving account etc.)
 - b. Initiate the Interledger Protocol Suite (ILP) transfer (ILP address of receiving account, amount, condition etc.)
 - c. Get a quote for the transfer.
3. The sender confirms the transfer and initiates it by way of the sending DFSP.

Design Considerations

1. There are a wide variety of identifiers that may be used to initiate the discovery process.
2. A lot of the data required is gathered during the Simple Payment Setup Protocol (SPSP).
3. The deployment scenarios will differ and as such the registries of identifiers will have different architectures (for example, distributed versus centralized).

Service Discovery vs Data Discovery

Rather than discovering data about the receiver, a more extensible solution uses the receiver identifier to resolve a service endpoint. The service at this endpoint can be standardized (this will be the SPSP entry-point), and in future this service may be extended to provide additional functionality and data.

By decoupling the discovery of the service from the service itself we define distinct phases in the preparation of an ILP transfer: *discovery* and *setup*. These phases can be defined by distinct entry and exit gates, and the specific implementations can be changed as long as the input and output of each phase is in a consistent format.

Using such an architecture, it is also possible to host the service at a URL that does not reveal any data about the receiver, allowing public discovery systems to be used without compromising the receiver's privacy.

Example of privacy protecting SPSP receiver URL:

- tel:+2612345678 - resolves to -> <https://ist.ng/ea472-cd5346-87df2h6680> (using a public unsecured registry)
- SPSP session initiated at <https://ist.ng/ea472-cd5346-87df2h6680> requires authentication

It is assumed that access to the SPSP receiver endpoints will be subject to policies that are designed to protect receiver data privacy (that is, either not exposed on the public internet or protected behind an effective authorization system).

Setup Phase

The setup phase is handled by SPSP. The protocol requires that some entity (which doesn't have to be the receiving DFSP) hosts a *receiver endpoint* at which the sender can query an SPSP Server for the data required to setup a transfer. All that is required to initiate setup using SPSP is the URL of the receiver endpoint.

Discovery Phase

Working back from the requirements to start the setup phase, it follows that the discovery phase must simply return a URL.

Normalization of Identifiers

Since the inputs to the discovery phase are only loosely typed as "an identifier" it may be useful to be specific and call this a URI.

Identifiers that do not have a natural URI form can usually be converted to one (or one can be defined for them). Where an identifier is provided to a sending system that is not a URI it is the responsibility of the sending system to determine the correct form based on the context and, if required, through interaction with the user.

Example 1

- Sender provides the identifier +26 78 097 8763
- Sending system recognizes this as an E164 format number and converts it to the URI tel:+26780978763.

Example 2

- Sender provides the identifier bob@dfsp1.com
- Sending system prompts the user to specify if this is an email address or an account identifier and then converts the identifier to the form mailto:bob@dfsp1.com or acct:bob@dfsp1.com.

This normalization allows a more rigid definition of a discovery service such that any service that accepts URIs and returns URLs could be used to resolve SPSP receiver endpoint URLs from receiver identifiers.

Discovery of SPSP Receiver Endpoint URL

Given all that has been defined to this point we can define a discovery service simply as a service that takes a URI representing a receiver identifier as input and returns a URL that should be the *receiver endpoint* for an SPSP server providing services for that receiver.

Sending systems (DFSPs) *should* determine which discovery service to use based on the URI scheme of the identifier.

The rules for this mapping (URI scheme/identifier type to discovery service) should be defined as part of each deployment. Mojaloop should provide implementations of one or more discovery services to bootstrap ecosystems where no such thing exists, but it should be possible for a DFSP to be configured to use other discovery services as long as they meet the minimum requirement of resolving a URL from a URI.

In deployments where all discovery is done through the same service (for example, a central directory) the logic for processing different identifier types can be deployed as part of that service. Therefore it will be unnecessary for the

sending system (DFSP) to be capable of calling different services based on the identifier type. While this is an optimization that may be possible for such a deployment, removing this logic from the DFSP will make introducing new discovery services in the future more difficult unless they are always proxied through the central service.

The logical steps for a sending DFSP are:

1. Get receiver identifier.
2. Normalize identifier to a URI if required.
3. Determine which discovery service to use based on URI scheme.
4. Resolve URI to URL using discovery service.
5. Initiate SPSP at resolved URL.

Design Considerations

The Mojaloop project has some specific design constraints and assumptions which drive the design of this implementation. The project favors the use of a central directory for discovery but also as a proxy for the quoting session with DFSPs. While this means it is not necessary for the discovery and setup to be decoupled, maintaining this architecture future-proofs the solution for deployments where these constraints and assumptions no longer hold.

Central Directory

The central directory will host a simple lookup service that resolves a receiver identifier to an SPSP URL. It should host different endpoints for each identifier type so that these can easily be changed in future if required and so the logic to differentiate between identifier types is built into the DFSPs from the start.

Example

- tel:+26123456789 -send-lookup-query-to-> https://ist.ng/api/tel
- acct:123456789@dfsp1.ng -send-lookup-query-to-> https://ist.ng/api/acct

The central directory lookup service will return a URL in the response for any identifier lookup. The URL is the one to use to initiate an SPSP session.

To optimize this process, in a deployment where the SPSP endpoints will be hosted by the IST, the URLs will use the same host (domain) as the lookup services. This will allow the client software at the sending DFSP to re-use the underlying connection for efficiency.

An HTTP session-based authorization model that is shared by both the lookup service and SPSP service will also mean that the client is able to re-use its authorized HTTP session further optimizing this process.

Using HTTP/2 in this architecture should further optimize this process.

It is expected that the individual account details will not be held at the IST but that the IST will provide an SPSP proxy service to the DFSPs blinded behind a non-descriptive SPSP endpoint URL.

Example

Step 1. Sender wishes to send money to +26 123 4567 (receiver identified by mobile number)

Step 2. Sending DFSP queries Central Directory for SPSP endpoint

```
GET /api/lookup/tel?identifier=tel%3A%2B261234567 HTTP/1.1
Host: ist.ng
```

Step 3. Central directory resolves identifier (account at DFSP1) and returns a local URL that is a proxy to the SPSP Server at DFSP1.

```
HTTP/1.1 200 Success
Content-Type: application/json
{
  "spspReceiver" : "https://ist.ng/api/spsp/
2911ca95-7bab-4699-b23a-6c64c03f3475"
}
```

Note: The URL gives nothing away about which DFSP is being proxied.

Step 4. Sending DFSP initiates SPSP session at <https://ist.ng/api/spsp/2911ca95-7bab-4699-b23a-6c64c03f3475>

Note: Both the lookup API and the SPSP endpoint are hosted at the IST.

User Retrieval Guide

Introduction

In this guide, we'll walk through the different steps of successfully retrieving a user from the Central Directory.

[Registering a user with the Central End User Registry](#) [Registering a DFSP with the Central Directory](#) [Looking up a user in the Central Directory](#) [Next Steps](#)

Step 1: Register a user with the Central End User Registry

Start off by registering a user with the url **http://dfsp1.com**. Simply provide the user's url and make a call to the register user endpoint. More detail about the response and errors can be found in the [Central End User Registry API documentation](#).

Request

```
POST http://central-end-user-registry/users
Content-Type: application/json
{
  "url": "http://user.dfsp.com"
}
```

Response

```
HTTP/1.1 201 Created
{
  "url": "http://user.dfsp.com",
}
```

```
    "number": "17141140"  
}
```

Step 2: Register a DFSP with the Central Directory

Once you have registered a user in the Central End User Registry, you need to register a DFSP with the Central Directory. This DFSP will simply be called **dfsp1**. A more in depth explanation can be found in the [API documentation](#).

Authorization

To register a DFSP with the directory, you need to authorize the call using [HTTP basic auth](#) with an admin key and secret.

Key	Secret
------------	---------------

admin admin

Request

```
POST http://central-directory/commands/register HTTP/1.1  
Content-Type: application/json  
Authorization: Basic YWRtaW46YWRtaW4=  
{  
    "name": "The first DFSP",  
    "shortName": "dfsp1",  
    "providerUrl": "http://url.com"  
}
```

Response

```
HTTP/1.1 201 CREATED  
Content-Type: application/json  
{  
    "name": "The first DFSP",  
    "shortName": "dfsp1",
```

```
"providerUrl": "http://url.com",
"key": "dfsp_key",
"secret": "dfsp_secret"
}
```

Step 3: Look up a user in the Central Directory

Now that you have created a user and registered a new DFSP, you can proceed with looking up the user in the Central Directory. Like before, more information can be found in the [API documentation](#).

Authorization

To retrieve the user from the directory, you need to authorize the call using [HTTP basic auth](#) with the key and secret returned from the register DFSP command above.

Key	Secret
dfsp1	dfsp1

Request

```
http://central-directory/resources/?identifierType=eur:17141140 HTTP/
1.1
Authorization: Basic ZGZzcDE6ZGZzcDE=
```

Response

```
HTTP/1.1 200 OK
[
  {
    "name": "The First DFSP",
    "providerUrl": "http://dfsp/users/1",
    "shortName": "dsfp1",
    "primary": "true",
    "registered": "true"
  }
]
```

Next Steps

Now that you have successfully created a DSFP and looked up a user, you should feel comfortable working with the other endpoints found in the [Central End User Registry API documentation](#) and the [Central Directory API documentation](#).

Central Directory API

In this guide, we'll walk through the different central directory endpoints: *POST Register a DFSP* *GET Get identifier types* *POST Register an identifier* *GET Lookup resource by identifier* *GET Get directory metadata* *GET Get authentication token* * *GET Directory health check*

The different endpoints often deal with these data structures: *Resource Object* *DFSP Object* *Identifier Type Object* *Metadata Object*

Information about various errors returned can be found here: * [Error Information](#)

Endpoints

Register a DFSP

This endpoint allows a DFSP to be registered to use the central directory.

HTTP Request

```
POST http://central-directory/commands/register
```

Authentication

Type	Description
HTTP	The username and password are the values of environment variables CDIR_ADMIN_KEY and
Basic	CDIR_ADMIN_SECRET. This is the same if Basic authentication or Token authentication are enabled.

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request body

Field	Type	Description
name	String	The name of the created DFSP
shortName	String	The shortName of the created DFSP
providerUrl	String	The url reference for the DFSP

Response 201 Created

Field	Type	Description
Object	DFSP	The <u>DFSP object</u> as saved

Request

```
POST http://central-directory/commands/register HTTP/1.1
Content-Type: application/json
{
  "name": "The first DFSP",
  "shortName": "dfsp1",
  "providerUrl": "http://url.com"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "name": "The first DFSP",
  "shortName": "dfsp1",
  "providerUrl": "http://url.com",
```

```

    "key": "dfsp_key",
    "secret": "dfsp_secret"
}

```

Errors (4xx)

Field	Description
-------	-------------

AlreadyExistsError The DFSP already exists (determined by name)

```
{
  "id": "AlreadyExistsError",
  "message": "The DFSP already exists (determined by name)"
}
```

Get identifier types

This endpoint allows retrieval of the identifier types supported by the central directory.

HTTP Request

```
GET http://central-directory/identifier-types
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1.
HTTP Bearer	If Token authentication enabled, the token must be passed in the Authorization header, as well as the name of the registered DFSP in the Directory-Api-Key header.

Response 200 OK

Field	Type	Description
Object	Array	List of supported <u>Identifier Type objects</u>

Request

```
GET http://central-directory/identifier-types HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
```

```
[  
  {  
    "identifierType": "eur",  
    "description": "Central end user registry"  
  }  
]
```

Register an identifier

This endpoint allows a DFSP to add an identifier associated with their account. When the identifier is retrieved from the [Lookup resource by identifier](#) endpoint, the url registered with the DFSP will be returned.

HTTP Request

```
POST http://central-directory/resources
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1.
HTTP Bearer	If Token authentication enabled, the token must be passed in the Authorization header, as well as the name of the registered DFSP in the Directory-Api-Key header.

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request body

Field	Type	Description
identifier	String	The identifier type and identifier to be created, separated by a colon
primary	String	(optional) Sets the DFSP as primary for the identifier, can either be <i>true</i> or <i>false</i> .

Primary will default to true if it is the first DFSP added for this identifier, and will default to false if another DFSP already has been added.

If the current DFSP being updated is primary and the primary value is set to false, an error will be thrown.

Response 201 Created

Field	Type	Description
Object	Resource	The newly-created <u>Resource object</u> as saved

Request

```
POST http://central-directory/resources HTTP/1.1
Content-Type: application/json
{
  "identifier": "eur:dfsp123",
  "primary": "true"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "name": "The First DFSP",
  "providerUrl": "http://dfsp/users/1",
  "shortName": "dsfp1",
  "primary": "true",
  "registered": "true"
}
```

Errors (4xx)

Field	Description
AlreadyExistsError	The identifier has already been registered by this DFSP { "id": "AlreadyExistsError", "message": "The identifier has already been registered by this DFSP" }

Lookup resource by identifier

This endpoint allows retrieval of a URI that will return customer information by supplying an identifier and identifier type.

HTTP Request

```
GET http://central-directory/resources?identifier={identifierType:identifier}
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1.
HTTP Bearer	If Token authentication enabled, the token must be passed in the Authorization header, as well as the name of the registered DFSP in the Directory-Api-Key header.

Query Params

Field	Type	Description
identifier	String	Valid identifier type and identifier separated with a colon

Response 200 OK

Field	Type	Description
Object	Array	An array of Resource objects retrieved

The returned array will contain one DFSP with primary set to true. All others should be set to false.

Request

```
GET http://central-directory/resources?identifier=eur:1234 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "name": "The First DFSP",
    "providerUrl": "http://dfsp/users/1",
    "shortName": "dsfp1",
    "primary": "true",
    "registered": "true"
  },
  {
    "name": "The Second DFSP",
    "providerUrl": "http://dfsp/users/2",
    "shortName": "dsfp2",
    "primary": "false",
    "registered": "false"
  }
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found.

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
```

```
}
```

Get directory metadata

Returns metadata associated with the directory

HTTP Request

```
GET http://central-directory
```

Response 200 OK

Field	Type	Description
Metadata	Object	The Metadata object for the directory

Request

```
GET http://central-directory HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "directory": "http://central-directory",
  "urls": {
    "health": "http://central-directory/health",
    "identifier_types": "http://central-directory/identifier-types",
    "resources": "http://central-directory/resources",
    "register_identifier": "http://central-directory/resources"
  }
}
```

Get authentication token

The get authentication endpoint generates an authentication token

HTTP Request

```
GET http://central-directory/auth_token
```

Authentication

Type	Description
HTTP Basic	The username and password are the key and secret of a registered DFSP, ex dfsp1:dfsp1.

Response 200 OK

Field	Type	Description
token	String	The generated authentication token

Request

```
GET http://central-directory/auth_token HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "token": "1234token4321"
}
```

Directory Health

Get the current status of the service

HTTP Request

```
GET http://central-directory/health
```

Response 200 OK

Field	Type	Description
status	String	The status of the directory, <i>OK</i> if the service is working

Request

```
GET http://central-directory/health HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "status": "OK"
}
```

Data Structures

Resource Object

A resource represents the information returned about an identifier and identifier type.

A resource object can have the following fields:

Name	Type	Description
name	String	Name of the DFSP
providerUrl	URI	A URI that can be called to get more information about the customer
shortName	String	Shortened name for the DFSP
primary	String	Details if the DFSP is set as primary, can either be <i>true</i> or <i>false</i>
registered	String	Returns <i>true</i> if DFSP is registered for the identifier, <i>false</i> if defaulted

DFSP Object

Represents a DFSP that has registered with the central directory.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. A DFSP object can have the following fields:

Name	Type	Description
name	String	The name of the created DFSP
shortName	String	The shortName of the created DFSP
providerUrl	String	The URL for the DFSP
key	String	Key used to authenticate with protected endpoints. Becomes the username for Basic Auth. Currently the same value as the name field
secret	String	Secret used to authenticate with protected endpoints. Currently the same value as the name field

Identifier Type Object

Represents an identifier type that is supported by the central directory.

An identifier type object can have the following fields:

Name	Type	Description
identifierType	String	Unique name of the identifier type
description	String	Description of the identifier type

Metadata Object

The central directory will return a metadata object about itself allowing client's to configure themselves properly.

A metadata object can have the following fields:

Name	Type	Description
directory	URI	The directory that generated the metadata
urls	Object	Paths to other methods exposed by this directory. Each field name is short name for a method and the value is the path to that method.

Error information

This section identifies the potential errors returned and the structure of the response.

An error object can have the following fields:

Name	Type	Description
id	String	An identifier for the type of error
message	String	A message describing the error that occurred
validationErrors	Array	<i>Optional</i> An array of validation errors
validationErrors[].message	String	A message describing the validation error
validationErrors[].params	Object	An object containing the field that caused the validation error
validationErrors[].params.key	String	The name of the field that caused the validation error
validationErrors[].params.value	String	The value that caused the validation error
validationErrors[].params.child	String	The name of the child field

```
HTTP/1.1 404 Not Found
Content-Type: application/json
{
  "id": "InvalidQueryParameterError",
  "message": "Error validating one or more query parameters",
  "validationErrors": [
    {
      "message": "'0' is not a registered identifierType",
      "params": {
        "key": "identifierType",
        "value": "0"
      }
    }
  ]
}
```

Directory Gateway

This project provides an API gateway to the Inter-System Trunk (IST) Directory Naming Service and provides resources to - "get metadata about a directory", "get customer information by providing identifier, identifierType", "Register a DFSP" and "get identifierTypes supported by the central directory".

Contents

- [Deployment](#)
- [Configuration](#)
- [API](#)
- [Logging](#)
- [Tests](#)

Deployment

Project is built using Maven and uses Circle for Continous Integration.

Installation and Setup

Anypoint Studio

- <https://www.mulesoft.com/platform/studio>
- Clone <https://github.com/mojaloop/interop-dfsp-directory.git> to local Git repository
- Import into Studio as a Maven-based Mule Project with pom.xml
- Go to Run -> Run As Configurations. Make sure interop-dfsp-directory project is highlighted.

Go to (x)=Arguments tab and make sure that -DMULE_ENV=dev is set in the VM Arguments.

Standalone Mule ESB

- <https://developer.mulesoft.com/download-mule-esb-runtime>
- Add the environment variable you are testing in (dev, prod, qa, etc). Open /conf/wrapper.conf and find the GC Settings section. Here there will be a series of wrapper.java.additional.(n) properties. create a new one after the last one where n=x (typically 14) and assign it the next number (i.e., wrapper.java.additional.15) and assign -DMULE_ENV=dev as its value (wrapper.java.additional.15=-DMULE_ENV=dev)
- Download the zipped project from Git
- Copy zipped file (Mule Archived Project) to /apps

Run Application

Anypoint Studio

- Run As Mule Application with Maven

Standalone Mule ESB

- CD to /bin -> in terminal type ./mule

Configuration

[pom.xml](#) and [circle.yml](#) can be found in the repo, also linked here

API

Below are the RAML and OpenAPI spec for reference

Mule console for the hosted service typically looks like this: `http://<host:port>/dfsp/directory/v1/console/`

- RAML [here](#)
- OpenAPI [here](#)

Logging

Server path to logs is: /logs/interop-dfsp-directory.log for example: /opt/mule/mule-dfsp1/logs/interop-dfsp-directory.log

Currently the logs are operational and include information such as TraceID and other details related to the calls or transactions such as path, method used, header information and payer/payee details.

Tests

Java Unit Tests exist for the project and include tests for:

- Invalid path should return 404
- Valid get user request should return valid response
- Valid add user request should return valide response
- Add valid account and retrieve it
- Add account without currency fails
- Add account without name fails
- Add account without account fails
- Add account with valid data

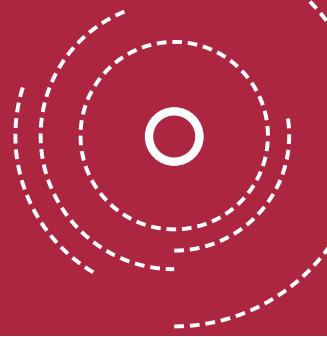
- Add account with invalid data fails
- Add account fails when presented with more fields

Anypoint Studio

- Run Unit Tests
- Test API with Anypoint Studio in APIKit Console
- Verify Responses in Studio Console output

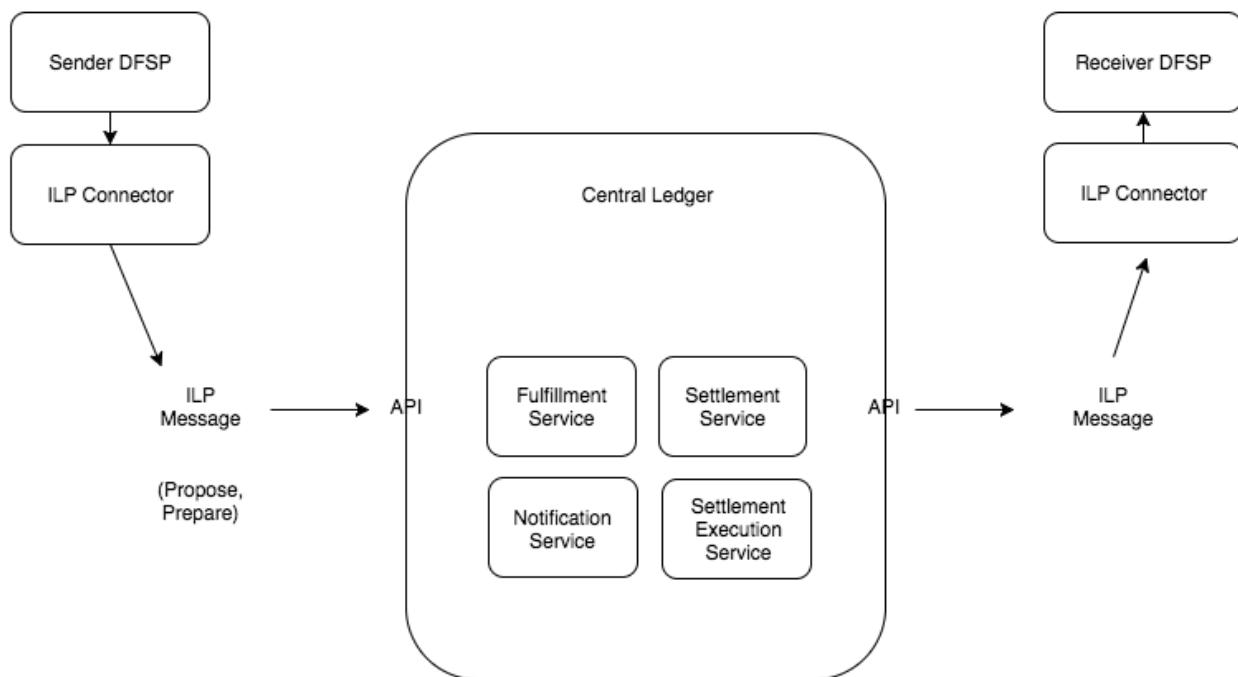
Tests are run as part of executing the Maven pom.xml as mvn clean package. Also, test can be run by running com.llp.interop.DirectoryFunctionalTest java class as JUnit Test.

Central Ledger



Central Ledger

Component Architecture



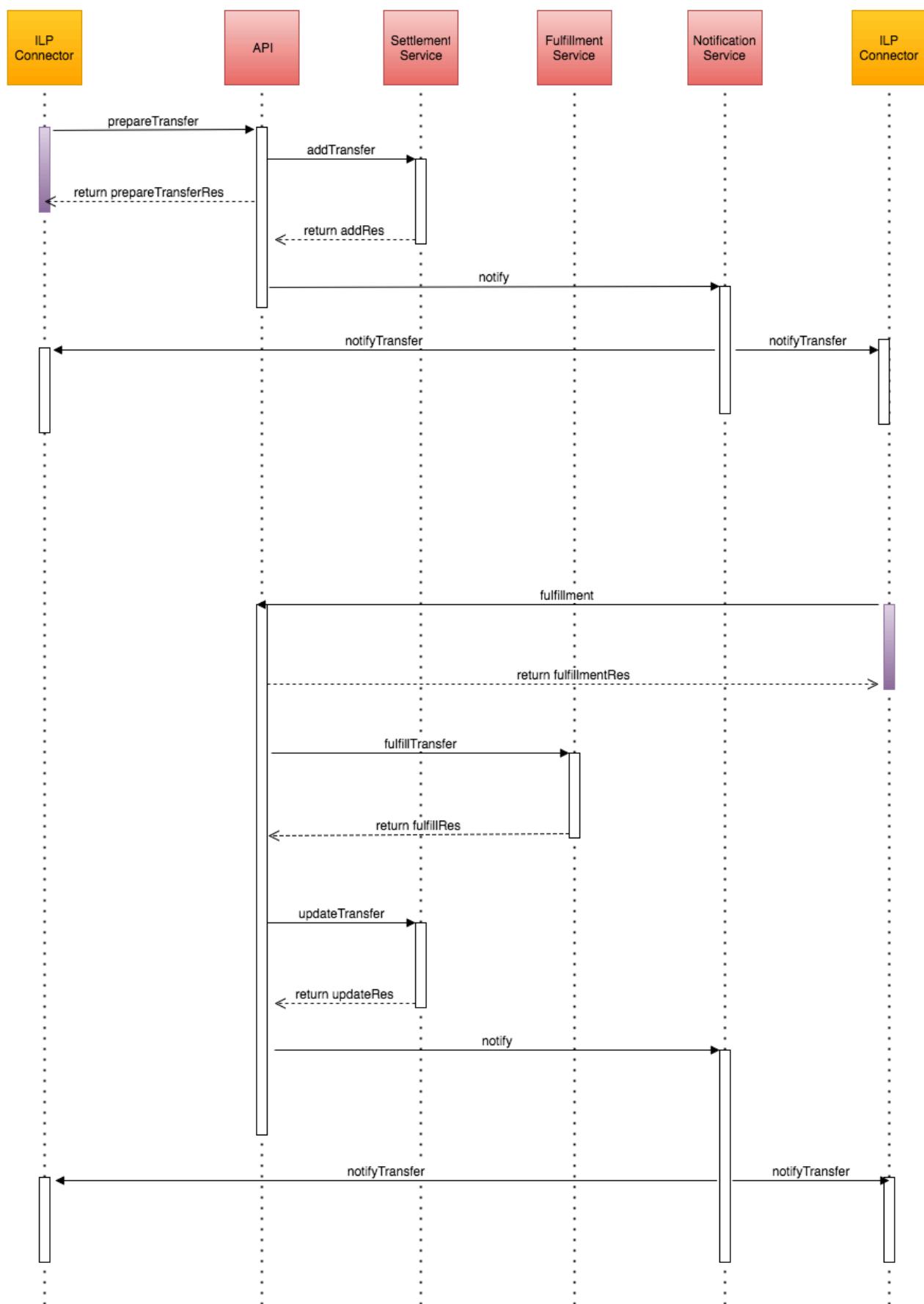
Component	Function	Interaction
API	Authenticate calling DFSPs; handle ILP calls; handle position calls	Calls internal services to handle fulfillment and settlement
Fulfillment Service	Handles validating ILP crypto conditions for transfers	Called by API; called by Settlement Service
Settlement Service	Handles recording fulfilled transfers for settlement; handles calculating DFSP positions; handles rebalance process and creation of settlement instructions	Called by API; called by Settlement Execution Service
Notification Service	Handles notifying DFSPs of transfers	Called by API; called by Fulfillment Service and Settlement Service
Settlement Execution Service	Execute settlement process to net cash positions	Called by API or internally based on scheme

Transfer/Fulfillment Flow

Sender DFSP

Central Ledger

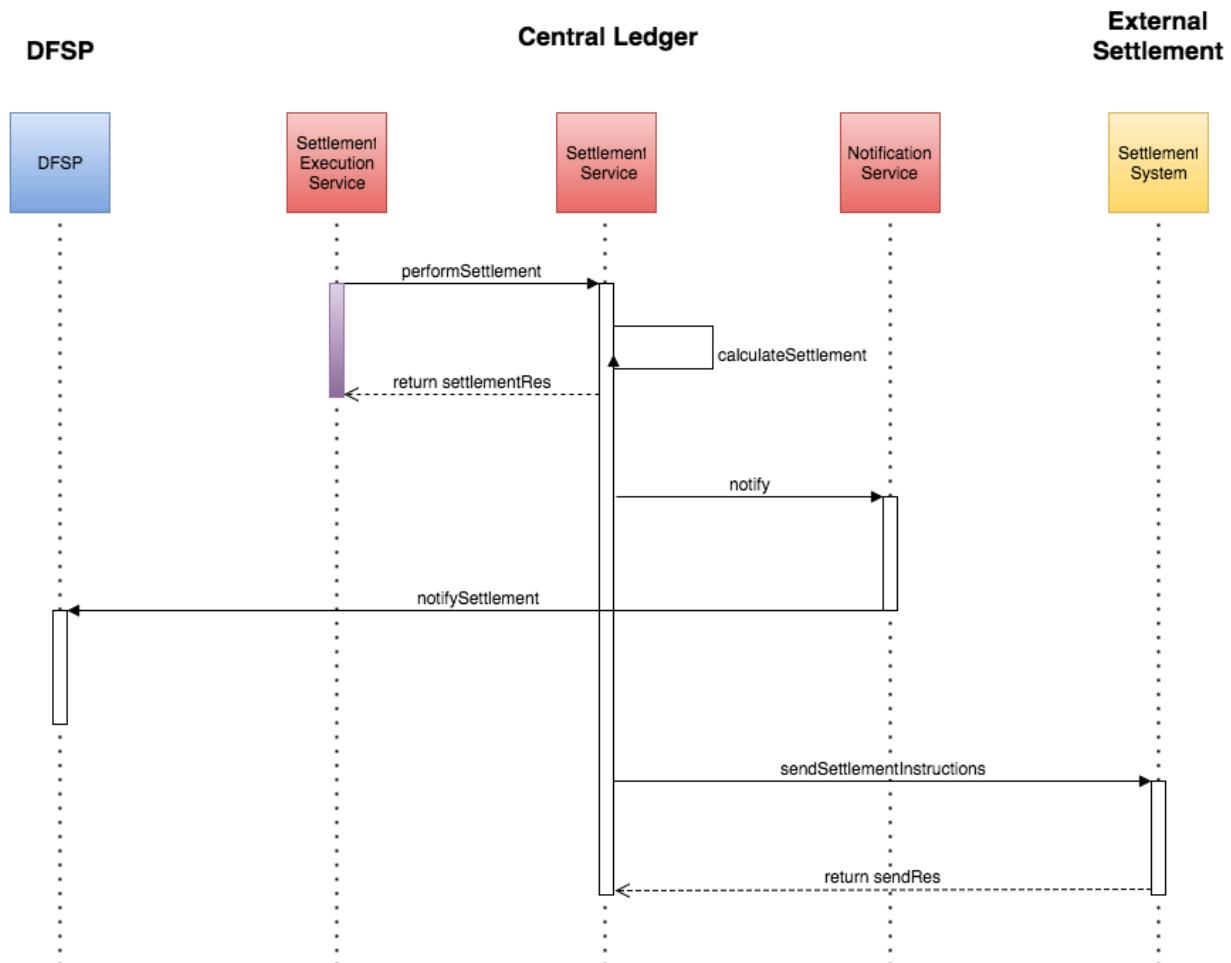
Receiver DFSP



Note

Only transfers where a Payer's currency is the same as a Payee's currency are operational in this release; i.e., even though multi-currency is supported, cross-currency is not supported at this time. The Cross-currency transfers use case is currently being investigated.

Settlement Flow



Endpoints

[Endpoints documentation](#)

Central Ledger API

The central ledger has two APIs targeted at different consumers. The DFSP API is used by DFSPs to prepare and execute transfers, and for DFSPs to retrieve their current settlement position. The Admin API is used by the operational hub to manage DFSPs and ensure the health of the system.

DFSP API endpoints

- POST [Create account](#)
- GET [Get account](#)
- PUT [Update account](#)
- PUT [Update account settlement](#)
- POST [Send message to account](#)
- GET [Get position for account](#)
- PUT [Prepare transfer](#)
- PUT [Fulfil transfer](#)
- PUT [Reject transfer](#)
- GET [Get transfer by id](#)
- GET [Get transfer fulfilment](#)
- GET [Get net positions](#)
- GET [Get metadata](#)
- POST [Get charge quote](#)
- POST [Get authentication token](#)
- GET [Health](#)

Admin API endpoints

- POST [Create account](#)
- PUT [Update admin account](#)

- GET [Get all accounts](#)
- POST [Create charge](#)
- PUT [Update charge](#)
- GET [Get all charges](#)
- GET [Get available permissions](#)
- POST [Create role](#)
- PUT [Update role](#)
- DELETE [Delete role](#)
- GET [Get all roles](#)
- POST [Get authentication token](#)
- POST [Create user](#)
- PUT [Update user](#)
- DELETE [Delete user](#)
- GET [Get user by id](#)
- GET [Get all users](#)
- GET [Get roles assigned to user](#)
- POST [Assign role to user](#)
- POST [Reject expired transfers](#)
- POST [Reject expired tokens](#)
- POST [Settle transfers and fees](#)
- GET [Health](#)

The API endpoints often deal with these [data structures](#):

- [Transfer Object](#)
- [Account Object](#)
- [Notification Object](#)
- [Metadata Object](#)
- [Position Object](#)
- [Charge Object](#)

Information about various errors returned can be found here: * [Error Information](#)

DFSP API

Create account

The create account endpoint will create an account in the ledger.

HTTP Request

```
POST http://central-ledger/accounts
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request Body

Field	Type	Description
Object	Account	An <u>Account object</u> to create

Response 201 Created

Field	Type	Description
Object	Account	The newly-created <u>Account object</u> as saved

Request

```
POST http://central-ledger/accounts HTTP/1.1
Content-Type: application/json
{
  "name": "dfsp1",
  "password": "dfsp1_password"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "id": "http://central-ledger/accounts/dfsp1",
  "name": "dfsp1",
  "created": "2017-01-03T19:50:39.744Z",
  "balance": "0",
  "is_disabled": false,
  "ledger": "http://central-ledger"
}
```

Errors (4xx)

Field	Description
RecordExistsError	The account already exists (determined by name)
{	<pre>"id": "RecordExistsError", "message": "The account has already been registered"</pre>

Get account

The get account endpoint will return information about the account. To successfully retrieve an account, make sure the [account has been previously created](#).

HTTP Request

```
GET http://central-ledger/accounts/dfsp1
```

URL Params

Field	Type	Description
name	String	The unique name for the account

Response 200 OK

Field	Type	Description
Object	Account	The Account object as saved

Request

```
GET http://central-ledger/accounts/dfsp1 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id": "http://central-ledger/accounts/dfsp1",
  "name": "dfsp1",
  "ledger": "http://central-ledger"
}
```

Response (Authenticated)

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id": "http://central-ledger/accounts/dfsp1",
  "name": "dfsp1",
  "created": "2016-09-28T17:03:37.168Z",
  "balance": 1000000,
  "is_disabled": false,
  "ledger": "http://central-ledger"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
```

```
  "id": "NotFoundError",
```

```
  "message": "The requested resource could not be found."
```

```
}
```

Update account

The update account endpoint will update the account's credentials and return the newly updated Account object.

HTTP Request

```
PUT http://central-ledger/accounts/dfsp1
```

URL Params

Field	Type	Description
name	String	The unique name for the account

Request body

Field	Type	Description
password	String	The new password for the account

Response 200 OK

Field	Type	Description
Object	Account	The <u>Account</u> object as saved

Request

```
PUT http://central-ledger/accounts/dfsp1 HTTP/1.1
Content-Type: application/json
{
  "password": "12345"
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name": "dfsp1",
  "id": "http://localhost:3000/accounts/dfsp1",
  "created": "2017-02-23T17:11:35.928Z",
```

```

    "is_disabled": true,
    "_links": {
      "self": "http://localhost:3000/accounts/dfsp1"
    }
}

```

Errors (4xx)

Field	Description
-------	-------------

NotFoundError The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Update account settlement

The update account settlement endpoint will create a new account settlement with the account's id and return the newly updated account settlement.

HTTP Request

```
PUT http://central-ledger/accounts/dfsp1/settlement
```

URL Params

Field	Type	Description
name	String	The name for the account

Request body

Field	Type	Description
account_number	String	The account number associated with the account's settlement
routing_number	String	The routing number associated with the account's settlement

Response 200 OK

Field	Type	Description
account_id	URI	The id for the account
account_number	String	The account number associated with the account's settlement
routing_number	String	The routing number associated with the account's settlement

Request

```
PUT http://central-ledger/accounts/dfsp1/settlement HTTP/1.1
Content-Type: application/json
{
  "account_number": "12345",
  "routing_number": "1234567891011",
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "account_id": "http://localhost:3000/accounts/dfsp1",
  "account_number": "12345",
  "routing_number": "1234567891011",
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found
	{ "id": "NotFoundError", "message": "The requested resource could not be found." }

Send message to account

The send messages endpoint posts messages to different accounts

HTTP Request

```
POST http://central-ledger/messages
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request Body

Field	Type	Description
ledger	URI	A link to the account's ledger
from	URI	A link to the from account
to	URI	A link to the to account
data	String	The data to be sent

Response 201 Created

Field	Type	Description
-------	------	-------------

Request

```
POST http://central-ledger/messages HTTP/1.1
Content-Type: application/json
{
  "ledger": "http://central-ledger",
  "from": "http://central-ledger/accounts/from",
  "to": "http://central-ledger/accounts/to",
  "data": { "foo": "bar" }
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
```

Errors (4xx)

Field	Description
RecordExistsError	The account already exists (determined by name) { "id": "RecordExistsError", "message": "The account has already been registered" }

Get position for account

The get account net positions endpoint returns the current net positions for the given account.

HTTP Request

```
GET http://central-ledger/positions/dfsp1
```

URL Params

Field	Type	Description
name	String	The unique name for the account

Response 200 OK

Field	Type	Description
account	String	Resource identifier
fees	Position	The Position object for an account's fees
transfers	Position	The Position object for an account's transfers
net	String	Net non-settled amount for the account as string

Request

```
GET http://central-ledger/positions/dsfp1 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "account": "http://localhost:3000/accounts/dfsp1",
  "fees": {
    "payments": "10",
    "receipts": "20",
    "net": "10"
  },
  "transfers": {
    "payments": "10",
    "receipts": "20",
    "net": "10"
  },
  "net": "20"
}
```

Errors (4xx)

Field	Description
UnprocessableEntityError	The provided entity is syntactically correct, but there is a generic semantic problem with it

```
{
  "id": "UnprocessableEntityError",
  "message": "The provided entity is syntactically correct, but there is a generic semantic problem with it"
}
```

Prepare transfer

The prepare transfer endpoint will create or update a transfer object. A transfer between two DFSPs must be prepared before it can be fulfilled. Before you can successfully prepare a transfer, make sure you have [created the corresponding accounts](#).

HTTP Request

```
PUT http://central-ledger/transfers/2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

URL Params

Field	Type	Description
id	String	A new UUID to identify this transfer

Request body

Field	Type	Description
Object	Transfer	A Transfer object to describe the transfer that should take place. For a conditional transfer, this includes an execution_condition

Response 201 Created

Field	Type	Description
Object	Transfer	The newly-created Transfer object as saved

Response 200 OK

Field	Type	Description
Object	Transfer	The updated Transfer object as saved

Request

```
PUT http://central-ledger/transfers/
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f HTTP/1.1
Content-Type: application/json
{
  "id": "http://central-ledger/transfers/
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f",
  "ledger": "http://central-ledger",
  "debits": [
    "account": "http://central-ledger/accounts/dfsp1",
    "amount": "50"
```

```

} ],
"credits": [
  "account": "http://central-ledger/accounts/dfsp2",
  "amount": "50"
],
"execution_condition": "ni:///sha-256;47DEQpj8HBSa-
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
"expires_at": "2016-12-26T00:00:01.000Z"
}

```

Response

```

HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "id": "http://central-ledger/transfers/
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f",
  "ledger": "http://central-ledger",
  "debits": [
    {
      "account": "http://central-ledger/accounts/dfsp1",
      "amount": 50
    }
  ],
  "credits": [
    {
      "account": "http://central-ledger/accounts/dfsp2",
      "amount": 50
    }
  ],
  "execution_condition": "ni:///sha-256;47DEQpj8HBSa-
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
  "expires_at": "2016-12-26T00:00:01.000Z",
  "state": "prepared",
  "timeline": {
    "prepared_at": "2017-01-03T16:16:18.958Z"
  }
}

```

Errors (4xx)

Field	Description
UnprocessableEntityError	The provided entity is syntactically correct, but there is a generic semantic problem with it

```
{
  "id": "UnprocessableEntityError",
  "message": "The provided entity is syntactically correct, but
there is a generic semantic problem with it"
}
```

Fulfil transfer

The fulfil transfer endpoint will either execute or cancel a transfer, depending on the existence of an *execution_condition* or *cancellation_condition*. To successfully fulfil a transfer, make sure the [transfer has previously been prepared](#).

HTTP Request

```
PUT http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204/
fulfilment
```

Headers

Field	Type	Description
Content-Type	String	Must be set to text/plain

URL Params

Field	Type	Description
id	String	Transfer UUID

Request Body

Field	Type	Description
Fulfilment	String	A fulfilment in string format

Response 200 OK

Field	Type	Description
Object	Transfer	The <u>Transfer object</u> as fulfilled

Request

```
PUT http://central-ledger/transfers/3a2a1d9e-8640-4d2d-
b06c-84f2cd613204/fulfilment HTTP/1.1
Content-Type: text/plain
oAKAAA
```

Response

```
HTTP/1.1 201 OK
Content-Type: application/json
{
  "id": "http://central-ledger/transfers/3a2a1d9e-8640-4d2d-
b06c-84f2cd613204",
  "ledger": "http://central-ledger",
  "debts": [
    {
      "memo": {
        "path": "blah",
        "interledger": "blah"
      },
      "amount": 50,
      "account": "http://central-ledger/accounts/dfsp1"
    }
  ],
  "credits": [
    {
      "memo": {
        "path": "blah",
        "interledger": "blah"
      },
      "amount": 50,
      "account": "http://central-ledger/accounts/dfsp2"
    }
  ],
  "execution_condition": "ni:///sha-256;47DEQpj8HBSa-
-TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
  "expires_at": "2016-12-26T00:00:01.000Z",
  "state": "executed",
  "timeline": {
    "prepared_at": "2016-12-19T16:04:01.316Z",
    "executed_at": "2016-12-19T16:04:55.766Z"
  }
}
```

Errors (4xx)

Field	Description
UnprocessableEntityError	The provided entity is syntactically correct, but there is a generic semantic problem with it
NotFoundError	The requested resource could not be found

```
{  
  "id": "NotFoundError",  
  "message": "The requested resource could not be found."  
}
```

Reject transfer

The reject transfer endpoint rejects the transfer with the given message. To successfully reject a transfer, make sure the [transfer has previously been prepared](#).

HTTP Request

```
PUT http://central-ledger/transfers/7d4f2a70-e0d6-42dc-9efb-6d23060ccd6f/  
rejection
```

URL Params

Field	Type	Description
id	String	Transfer UUID

Request Body

Field	Type	Description
Rejection	String	The rejection message in string format

Response 200 OK

Field	Type	Description
Object	Transfer	The <u>Transfer object</u> as rejected

Request

```
PUT http://central-ledger/transfers/  
7d4f2a70-e0d6-42dc-9efb-6d23060ccd6f/rejection HTTP/1.1  
Content-Type: text/plain  
this transfer is bad
```

Response

```
HTTP/1.1 200 OK  
{  
  "id": "http://central-ledger/transfers/  
7d4f2a70-e0d6-42dc-9efb-6d23060ccd6f",  
  "ledger": "http://central-ledger",  
  "debits": [  
    {  
      "memo": {  
        "path": "blah",  
        "interledger": "blah"  
      },  
      "amount": 50,  
      "account": "http://central-ledger/accounts/dfsp1"  
    }  
,  
  "credits": [  
    {  
      "memo": {  
        "path": "blah",  
        "interledger": "blah"  
      },  
      "amount": 50,  
      "account": "http://central-ledger/accounts/dfsp2"  
    }  
,  
  "execution_condition": "ni:///sha-256;47DEQpj8HBSa-  
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",  
  "expires_at": "2016-12-26T00:00:01.000Z",  
  "state": "rejected",  
  "timeline": {  
    "prepared_at": "2017-01-03T16:16:18.958Z",  
    "rejected_at": "2017-01-03T19:58:42.100Z"  
  },  
  "rejection_reason": "this transfer is bad"  
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found { "id": "UnpreparedTransferError", "message": "The provided entity is syntactically correct, but there is a generic semantic problem with it." }

Get transfer by id

HTTP Request

```
GET http://central-ledger/transfers/2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f
```

URL Params

Field	Type	Description
id	String	Transfer UUID

Response 200 OK

Field	Type	Description
Object	Transfer	The <u>Transfer object</u> as saved

Request

```
GET http://central-ledger/transfers/  
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6fHTTP/1.1
```

Response

```
HTTP/1.1 200 OK  
{  
  "id": "http://central-ledger/transfers/  
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f",  
  "ledger": "http://central-ledger",
```

```

"debits": [
  {
    "account": "http://central-ledger/accounts/dfsp1",
    "amount": "50.00",
    "memo": "{\"path\":\"blah\", \"interledger\":\"blah\"}"
  }
],
"credits": [
  {
    "account": "http://central-ledger/accounts/dfsp2",
    "amount": "50.00",
    "memo": "{\"path\":\"blah\", \"interledger\":\"blah\"}"
  }
],
"execution_condition": "ni:///sha-256;47DEQpj8HBSa-
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
"expires_at": "2016-12-26T00:00:01.000Z",
"state": "executed",
"timeline": {
  "prepared_at": "2016-12-19T16:04:01.316Z",
  "executed_at": "2016-12-19T16:04:55.766Z"
},
"rejection_reason": null
}

```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Get transfer fulfilment

The get transfer fulfilment endpoint will return the fulfilment for a transfer that has been executed or cancelled. To successfully retrieve a transfer fulfilment, make sure the [transfer has previously been fulfilled](#).

HTTP Request

```
GET http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204/
```

fulfilment

URL Params

Field	Type	Description
id	String	Transfer UUID

Response 200 OK

Field	Type	Description
Fulfilment	String	The fulfilment for the transfer

Request

```
GET http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204/fulfilment HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
oAKAAA
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found
{ "id": "NotFoundError", "message": "The requested resource could not be found." }	

Get net positions

The get current net positions endpoint returns the current net positions for all accounts in the ledger.

HTTP Request

```
GET http://central-ledger/positions
```

Response 200 OK

Field	Type	Description
positions	Array	List of current Position objects for the ledger

Request

```
GET http://central-ledger/positions HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "positions": [
    {
      "account": "http://central-ledger/accounts/dfsp1",
      "payments": "0",
      "receipts": "0",
      "net": "0"
    },
    {
      "account": "http://central-ledger/accounts/dfsp2",
      "payments": "100",
      "receipts": "0",
      "net": "-100"
    },
    {
      "account": "http://central-ledger/accounts/dfsp3",
      "payments": "0",
      "receipts": "100",
      "net": "100"
    }
  ]
}
```

Get net positions for account

The get current net positions endpoint returns the current net positions for all accounts in the ledger.

HTTP Request

```
GET http://central-ledger/positions/dfsp1
```

Response 200 OK

Field	Type	Description
Object	Array	<u>Position objects</u> for the account

Request

```
GET http://central-ledger/positions/dfsp1 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "account": "http://localhost:3000/accounts/dfsp1",
  "fees": {
    "payments": "4",
    "receipts": "0",
    "net": "-4"
  },
  "transfers": {
    "payments": "40",
    "receipts": "0",
    "net": "-40"
  },
  "net": "-44"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Get metadata

The get metadata endpoint returns metadata associated with the ledger.

HTTP Request

```
GET http://central-ledger
```

Response 200 OK

Field	Type	Description
Metadata	Object	The Metadata object for the ledger

Request

```
GET http://central-ledger HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "currency_code": null,
  "currency_symbol": null,
  "ledger": "http://central-ledger",
  "urls": {
    "auth_token": "http://central-ledger/auth_token",
    "health": "http://central-ledger/health",
    "positions": "http://central-ledger/positions",
    "account": "http://central-ledger/accounts/:name",
    "accounts": "http://central-ledger/accounts",
    "send_message": "http://central-ledger/messages",
    "transfer": "http://central-ledger/transfers/:id",
    "transfer_fulfillment": "http://central-ledger/transfers/:id/
fulfilment",
    "transfer_rejection": "http://central-ledger/transfers/:id/
rejection",
    "notifications": "ws://central-ledger/websocket"
  },
  "precision": 10,
  "scale": 2
}
```

Get a charge quote

Get a list of charge quotes for a given amount, that the sender would be responsible for paying

HTTP Request

```
POST http://central-ledger/charges/quote
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request Body

Field	Type	Description
amount	Decimal	The amount for quote

Response 200 OK

Field	Type	Description
N/A	Array	A list of charge quotes

Request

```
POST http://central-ledger/charges/quotes HTTP/1.1
Content-Type: application/json
{
  "amount": "10.00"
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
[
  {
    "name": "charge1",
    "charge_type": "fee",
```

```

    "code": "001",
    "amount": "0.25",
    "currency_code": "USD",
    "currency_symbol": "$"
},
{
    "name": "charge2",
    "charge_type": "fee",
    "code": "002",
    "amount": "2.00",
    "currency_code": "USD",
    "currency_symbol": "$"
}
]

```

Errors (4xx)

Field	Description
-------	-------------

InvalidBodyError Body does not match schema

```
{
  "id": "InvalidBodyError",
  "message": "Body does not match schema"
}
```

Get authentication token

The get authentication endpoint generates an authentication token

HTTP Request

```
GET http://central-ledger/auth_token
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Basic	Defaults to admin:admin

Response 200 OK

Field	Type	Description
token	String	The generated authentication token

Request

```
GET http://central-ledger/auth_token HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "token": "1234token4321"
}
```

Health

Get the current status of the service

HTTP Request

```
GET http://central-ledger/health
```

Response 200 OK

Field	Type	Description
status	String	The status of the ledger, <i>OK</i> if the service is working

Request

```
GET http://central-ledger/health HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "status": "OK"
}
```

Admin API

Create account admin

The create account endpoint will create an account in the ledger.

HTTP Request

```
POST http://central-ledger/accounts
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

Request Body

Field	Type	Description
Object	Account	An Account object to create

Response 201 Created

Field	Type	Description
Object	Account	The newly-created Account object as saved

Request

```
POST http://central-ledger/accounts HTTP/1.1
Content-Type: application/json
{
  "name": "dfspl1",
  "password": "dfspl_password"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "id": "http://central-ledger/accounts/dfsp1",
  "name": "dfsp1",
  "created": "2017-01-03T19:50:39.744Z",
  "balance": "0",
  "is_disabled": false,
  "ledger": "http://central-ledger"
}
```

Errors (4xx)

Field	Description
RecordExistsError	The account already exists (determined by name)
{	<pre>"id": "RecordExistsError", "message": "The account has already been registered"</pre>

Get account admin

The get account endpoint will return information about the account. To successfully retrieve an account, make sure the [account has been previously created](#).

HTTP Request

```
GET http://central-ledger-admin/accounts/dfsp1
```

URL Params

Field	Type	Description
name	String	The unique name for the account

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Account	The Account object as saved

Request

```
GET http://central-ledger/accounts/dfsp1 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id": "http://central-ledger/accounts/dfsp1",
  "name": "dfsp1",
  "created": "2016-09-28T17:03:37.168Z",
  "balance": 1000000,
  "is_disabled": false,
  "ledger": "http://central-ledger"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Update admin account

The update admin account endpoint will update the account's 'is_disabled' field and return the newly updated Account

object.

HTTP Request

```
PUT http://central-ledger-admin/accounts/dfsp1
```

URL Params

Field	Type	Description
name	String	The unique name for the account

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Account	The <u>Account object</u> as saved

Request

```
PUT http://central-ledger-admin/accounts/dfsp1 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "name": "dfsp1",
  "id": "http://localhost:3000/accounts/dfsp1",
  "created": "2017-02-23T17:11:35.928Z",
  "is_disabled": true,
  "_links": {
    "self": "http://localhost:3000/accounts/dfsp1"
  }
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found { "id": "NotFoundError", "message": "The requested resource could not be found." }

Get all accounts

The get all accounts endpoint lists all created accounts

HTTP Request

```
GET http://central-ledger-admin/accounts
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	An array of all created <u>Account objects</u>

Request

```
GET http://central-ledger-admin/accounts HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "id": "http://central-ledger/accounts/dfsp1",
```

```

    "name": "dfsp1",
    "created": "2016-09-28T17:03:37.168Z",
    "balance": 1000000,
    "is_disabled": false,
    "ledger": "http://central-ledger"
},
{
    "id": "http://central-ledger/accounts/dfsp2",
    "name": "dfsp2",
    "created": "2016-09-28T17:03:37.168Z",
    "balance": 1000000,
    "is_disabled": false,
    "ledger": "http://central-ledger"
}
]

```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Create charge

Create a charge that will be applied across the dfsp on transfer execution

HTTP Request

```
POST http://central-ledger-admin/charges
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
name	String	Unique name for the charge
charge_type	String	Type of the charge, should be <i>fee</i>
rate_type	String	How the charge rate is applied, either <i>flat</i> or <i>percent</i>
rate	String	Charge rate, represented as a decimal for percent (<i>5% is 0.05</i>) and as the actual value for flat
minimum	String	Minimum transfer amount needed to apply the charge
maximum	String	Maximum transfer amount needed to apply the charge
code	String	Three character code used to identify the charge
is_active	Boolean	Set by admin, determines whether charge should be applied or not
payer	String	Account that pays the fee generated by the charge, either <i>sender</i> , <i>receiver</i> , or <i>ledger</i>
payee	String	Account that receives the fee generated by the charge, either <i>sender</i> , <i>receiver</i> , or <i>ledger</i>

Response 200 OK

Field	Type	Description
Object	Charge	Newly created charge

Request

```
POST http://central-ledger-admin/charges HTTP/1.1
Content-Type: application/json
{
  "name": "charge",
  "charge_type": "fee",
  "rate_type": "flat",
  "rate": "1.00",
  "minimum": "0.00",
  "maximum": "100.00",
  "code": "001",
  "is_active": true,
  "payer": "sender",
  "payee": "receiver"
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
  "name": "charge",
  "id": 5,
  "charge_type": "fee",
  "rate_type": "flat",
  "rate": "1.00",
  "minimum": "0.00",
  "maximum": "100.00",
  "code": "001",
  "is_active": true,
  "created": "2017-03-10T17:56:35.966Z",
  "payer": "sender",
  "payee": "receiver"
}
```

Errors (4xx)

Field	Description
InvalidBodyError	Body does not match schema.
ValidationError	Payer and payee should be set to 'sender', 'receiver', or 'ledger' and should not have the same value.
RecordExistsError	The charge has already been created.

```
{
  "id": "InvalidBodyError",
  "message": "Body does not match schema"
}
```

Update charge

Update an existing charge, only the name, charge type, minimum, maximum, code, and is active fields may be updated.

HTTP Request

```
PUT http://central-ledger-admin/charges/charge_name
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

URL Params

Field	Type	Description
name	String	The unique name for the account

Request Body

Field	Type	Description
name	String	Unique name for the charge
charge_type	String	Type of the charge, should be <i>fee</i>
minimum	String	Minimum transfer amount needed to apply the charge
maximum	String	Maximum transfer amount needed to apply the charge
code	String	Three character code used to identify the charge
is_active	Boolean	Set by admin, determines whether charge should be applied or not

Response 200 OK

Field	Type	Description
Object	Charge	Newly updated charge

Request

```
PUT http://central-ledger-admin/charges/charge_name HTTP/1.1
Content-Type: application/json
{
  "name": "updated_charge_name",
  "charge_type": "fee",
  "minimum": "0.01",
  "maximum": "100.01",
  "code": "002",
```

```
        "is_active": false  
    }
```

Response

```
HTTP/1.1 200 OK  
Content-Type: application/json  
{  
    "name": "updated_charge_name",  
    "id": 5,  
    "charge_type": "fee",  
    "rate_type": "flat",  
    "rate": "1.00",  
    "minimum": "0.01",  
    "maximum": "100.01",  
    "code": "002",  
    "is_active": false,  
    "created": "2017-03-10T17:56:35.966Z",  
    "payer": "sender",  
    "payee": "receiver"  
}
```

Errors (4xx)

Field	Description
InvalidBodyError	Body does not match schema.
ValidationError	Payer and payee should be set to 'sender', 'receiver', or 'ledger' and should not have the same value.
RecordExistsError	The charge has already been created.

```
{  
    "id": "InvalidBodyError",  
    "message": "Body does not match schema"  
}
```

Get all charges

The get all charges endpoint lists all created charges

HTTP Request

```
GET http://central-ledger-admin/charges
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	An array of all created Charge objects

Request

```
GET http://central-ledger-admin/charges HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "name": "fee_1",
    "id": 1,
    "charge_type": "fee",
    "rate_type": "flat",
    "rate": "1.00",
    "minimum": "0.00",
    "maximum": "15.00",
    "code": "001",
    "is_active": true,
    "created": "2017-04-10T19:49:54.850Z",
    "payer": "sender",
    "payee": "receiver"
  }
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found <pre>{ "id": "NotFoundError", "message": "The requested resource could not be found." }</pre>

Get available permissions

The get available permissions endpoint lists all available permissions

HTTP Request

```
GET http://central-ledger-admin/permissions
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	An array of <u>Permission objects</u>

Request

```
GET http://central-ledger-admin/permissions HTTP/1.1
```

Response

```
HTTP/1.1 200 OK  
[  
  {  
    "key": "ACCOUNTS_CREATE",
```

```

        "description": "Create an account"
    },
{
    "key": "ACCOUNTS_LIST",
    "description": "List all accounts"
}
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Create role

The create role endpoint will create a user role.

HTTP Request

```
POST http://central-ledger-admin/roles
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
name	String	The name of the role
description	String	The description of the role
permissions	Array	An array of Permission object keys

Response 201 Created

Field	Type	Description
Object	Role	The newly-created <u>Role object</u> as saved

Request

```
POST http://central-ledge-adminr/roles HTTP/1.1
Content-Type: application/json
{
    "name" : "Create role.",
    "description" : "An admin role for creating and listing users.",
    "permissions" : ["ACCOUNTS_CREATE, ACCOUNTS_LIST"]
}
```

Response

```
HTTP/1.1 201 CREATED
Content-Type: application/json
{
    "roleId": "374885fd-2384-429c-9355-57466aff2dd2",
    "name": "Create role.",
    "description": "An admin role for creating and listing users.",
    "permissions": [
        "ACCOUNTS_CREATE, ACCOUNTS_LIST"
    ]
}
```

Errors (4xx)

Field	Description
RecordExistsError	The role already exists (determined by name)

```
{
    "id": "RecordExistsError",
    "message": "The role has already been created"
}
```

Update role

The update role endpoint will update a given user role.

HTTP Request

```
PUT http://central-ledger-admin/roles/374885fd-2384-429c-9355-57466aff2dd2
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
name	String	The name of the role
description	String	The description of the role
permissions	Array	An array of <u>Permission object</u> keys

Response 200 OK

Field	Type	Description
Object	Role	The newly-created <u>Role object</u> as saved

Request

```
PUT http://central-ledger-admin/roles/  
374885fd-2384-429c-9355-57466aff2dd2 HTTP/1.1  
Content-Type: application/json  
{  
    "name" : "Create role.",  
    "description" : "An admin role for creating, listing, and  
deleting users.",  
    "permissions" : ["ACCOUNTS_CREATE, ACCOUNTS_LIST,  
ACCOUNT_DELETE"]  
}
```

Response

```
HTTP/1.1 200 OK  
Content-Type: application/json
```

```
{
  "roleId": "374885fd-2384-429c-9355-57466aff2dd2",
  "name": "Create role.",
  "description": "An admin role for creating and listing, and deleting users.",
  "permissions": [
    "ACCOUNTS_CREATE, ACCOUNTS_LIST, ACCOUNT_DELETE"
  ]
}
```

Errors (4xx)

Field	Description
-------	-------------

NotFoundError The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Delete role

The delete role endpoint will delete a given user role.

HTTP Request

```
DELETE http://central-ledger-admin/roles/374885fd-2384-429c-9355-57466aff2dd2
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Response 204 No Content

Field	Type	Description
-------	------	-------------

Request

```
DELETE http://central-ledger-adminr/roles/  
374885fd-2384-429c-9355-57466aff2dd2 HTTP/1.1  
Content-Type: application/json
```

Response

```
HTTP/1.1 204 NO CONTENT  
Content-Type: application/json
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found
{ "id": "NotFoundError", "message": "The requested resource could not be found." }	

Get all roles

The get all roles endpoint returns a list of all roles for the central ledger

HTTP Request

```
GET http://central-ledger-admin/roles
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	List of <u>Role objects</u> for the central ledger

Request

```
GET http://central-ledger-admin/roles HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
[
  {
    "roleId": "7d3bdc7a-2f83-456f-b174-3ad1f477c7be",
    "name": "Account Manager",
    "description": "Create a view accounts.",
    "permissions": [
      "ACCOUNTS_CREATE",
      "ACCOUNTS_LIST"
    ]
  },
  {
    "roleId": "7d3bdc7a-2f54-456f-b174-3ad1f477c7b2",
    "name": "User Manager",
    "description": "Create a view users",
    "permissions": [
      "USERS_CREATE",
      "USERS_LIST"
    ]
  }
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Get admin authentication token

The get admin authentication token endpoint generates an admin authentication token

HTTP Request

```
POST http://central-ledger-admin/auth_token
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
token	String	The generated authentication token

Request

```
POST http://central-ledger-admin/auth_token HTTP/1.1
Content-Type: application/json
{
  "key": "login_key"
}
```

Response

```
HTTP/1.1 200 OK
{
  "token": "1234token4321"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Create user

The create user endpoint will create an admin user.

HTTP Request

```
POST http://central-ledger-admin/users
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
Object	User	A <u>User object</u> to create

Response 201 Created

Field	Type	Description
Object	User	The newly-created <u>User object</u> as saved

Request

```
POST http://central-ledger-admin/users HTTP/1.1
Content-Type: application/json
{
  "firstName": "First",
  "lastName": "Last",
  "email": "email@central-ledger.com",
  "key": "login_key"
}
```

Response

```
HTTP/1.1 201 CREATED
```

```
Content-Type: application/json
{
  "userId": "e181fc02",
  "key": "key",
  "lastName": "Last",
  "firstName": "First",
  "email": "email@central-ledger.com",
  "isActive": true,
  "createdDate": "2017-04-10T21:15:06.378Z"
}
```

Errors (4xx)

Field	Description
RecordExistsError	The account already exists (determined by name)
{ "id": "RecordExistsError", "message": "The account has already been registered" }	

Update user

The update user endpoint will update the give user.

HTTP Request

```
PUT http://central-ledger-admin/users/e181fc02
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
Object	User	A <u>User object</u> to update

Response 200 OK

Field	Type	Description
Object	User	The update User object as saved

Request

```
PUT http://central-ledger-admin/users HTTP/1.1
Content-Type: application/json
{
  "firstName": "FirstA",
  "lastName": "LastB",
  "email": "email@central-ledger.com",
  "key": "login_key"
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "userId": "e181fc02",
  "key": "key",
  "lastName": "LastB",
  "firstName": "FirstA",
  "email": "email@central-ledger.com",
  "isActive": true,
  "createdDate": "2017-04-10T21:15:06.378Z"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Delete user

The get user by id endpoint returns an admin user with the given id

HTTP Request

```
DELETE http://central-ledger-admin/users/e181fc02
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description

Request

```
DELETE http://central-ledger-admin/users/e181fc02 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{ }
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Get user by id

The get user by id endpoint returns an admin user with the given id

HTTP Request

```
GET http://central-ledger-admin/users/e181fc02
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	User	<u>User object</u> for the given id

Request

```
GET http://central-ledger-admin/users/e181fc02 HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "userId": "e181fc02",
  "key": "key",
  "lastName": "lastName",
  "firstName": "firstName",
  "email": "email@central-ledger.com",
  "isActive": true,
  "createdDate": "2017-04-10T21:15:06.378Z"
}
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found <pre>{ "id": "NotFoundError", "message": "The requested resource could not be found." }</pre>

Get all users

The get all users endpoint returns all admin users for the central ledger

HTTP Request

```
GET http://central-ledger-admin/users
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	A list of <u>User objects</u> for the central ledger

Request

```
GET http://central-ledger-admin/users HTTP/1.1
```

Response

```
HTTP/1.1 200 OK  
[  
  {  
    "userId": "e181fc02",
```

```

    "key": "key",
    "lastName": "lastName",
    "firstName": "firstName",
    "email": "email@central-ledger.com",
    "isActive": true,
    "createdDate": "2017-04-10T21:15:06.378Z"
},
{
    "userId": "e181fc03",
    "key": "key",
    "lastName": "lastName2",
    "firstName": "firstName2",
    "email": "email2@central-ledger.com",
    "isActive": false,
    "createdDate": "2017-04-10T21:15:06.378Z"
}
]

```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found
{ "id": "NotFoundError", "message": "The requested resource could not be found." }	

Get roles assigned to user

This endpoint returns a list of a user's roles

HTTP Request

```
GET http://central-ledger-admin/users/e181fc02/roles
```

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	An array of Role objects for the account

Request

```
GET http://central-ledger-admin/users/e181fc02/roles HTTP/1.1
```

Response

HTTP/1.1 200 OK

```
[  
  {  
    "roleId": "7d3bdc7a-2f83-456f-b174-3ad1f477c7be",  
    "name": "Create account",  
    "description": "Role for creating and listing users",  
    "permissions": [  
      "ACCOUNTS_CREATE",  
      "ACCOUNTS_LIST"  
    ]  
  }  
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found { "id": "NotFoundError", "message": "The requested resource could not be found." }

Assign role to user

This endpoint assigns a role to an admin user

HTTP Request

```
POST http://central-ledger/users/e181fc02/roles
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Request Body

Field	Type	Description
Object	Array	An array of role ids

Response 200 OK

Field	Type	Description
Object	Array	An array of <u>Role objects</u> for the account

Request

```
POST http://central-ledger/accounts HTTP/1.1
Content-Type: application/json
["7d3bdc7a-2f83-456f-b174-3ad1f477c7be", "7d3bdc7a-2f83-456f-
b174-3ad1f477c7cf"]
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
[
  {
    "roleId": "7d3bdc7a-2f83-456f-b174-3ad1f477c7be",
    "name": "Create account",
    "description": "Role for creating and listing users",
    "permissions": [
      "ACCOUNTS_CREATE",
      "ACCOUNTS_LIST"
    ]
  }
```

```
    }
]
```

Errors (4xx)

Field	Description
NotFoundError	The requested resource could not be found

```
{
  "id": "NotFoundError",
  "message": "The requested resource could not be found."
}
```

Reject expired transfers

This endpoint rejects all expired transfers

HTTP Request

```
POST http://central-ledger-admin/webhooks/reject-expired-transfers
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	A list of rejected transfer ids

Request

```
POST http://central-ledger-admin/webhooks/reject-expired-transfers
HTTP/1.1
Content-Type: application/json
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
[12, 13, 14]
```

Reject expired tokens

This endpoint rejects all expired tokens

HTTP Request

```
POST http://central-ledger-admin/webhooks/reject-expired-tokens
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
Object	Array	A list of rejected tokens

Request

```
POST http://central-ledger-admin/webhooks/reject-expired-transfers
HTTP/1.1
Content-Type: application/json
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
["123-abc", "456-def", "789-ghi"]
```

Settle transfers and fees

The settle transfers and fees endpoint settles all unsettled transfers and fees

HTTP Request

```
POST http://central-ledger-admin/webhooks/settle-transfers
```

Headers

Field	Type	Description
Content-Type	String	Must be set to application/json
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
transfers	Array	A list of settled transfer ids
fees	Array	A list of settled fee ids

Request

```
POST http://central-ledger-admin/webhooks/reject-expired-transfers  
HTTP/1.1  
Content-Type: application/json
```

Response

```
HTTP/1.1 200 OK  
Content-Type: application/json  
{  
  "transfers": [  
    {  
      "source": {  
        "account_number": "1234",  
        "routing_number": "5678"  
      },  
      "destination": {  
        "account_number": "4321",  
      }  
    }  
  ]  
}
```

```

        "routing_number": "8765"
    },
    "amount": {
        "currency_code": "$",
        "value": "20.00",
        "description": "dfsp2"
    }
}
],
"fees": [
{
    "source": {
        "account_number": "1234",
        "routing_number": "5678"
    },
    "destination": {
        "account_number": "4321",
        "routing_number": "8765"
    },
    "amount": {
        "currency_code": "$",
        "value": "3",
        "description": "dfsp2"
    }
}
]
}

```

Admin health

Get the current status of the admin service

HTTP Request

GET `http://central-ledger-admin/health`

Headers

Field	Type	Description
Authorization	Bearer Token	JWT based access token

Response 200 OK

Field	Type	Description
status	String	The status of the ledger, <i>OK</i> if the service is working

Request

```
GET http://central-ledger-admin/health HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
  "status": "OK"
}
```

Data Structures

Transfer Object

A transfer represents money being moved between two DFSP accounts at the central ledger.

The transfer must specify an execution_condition, in which case it executes automatically when presented with the fulfilment for the condition. (Assuming the transfer has not expired or been canceled first.) Currently, the central ledger only supports the condition type of PREIMAGE-SHA-256 and a max fulfilment length of 65535.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. A transfer object can have the following fields:

Name	Type	Description
id	URI	Resource identifier
ledger	URI	The ledger where the transfer will take place
debits	Array	Funds that go into the transfer
debits[].account	URI	Account holding the funds
debits[].amount	String	Amount as decimal

Name	Type	Description
debts[] invoice	URI	<i>Optional</i> Unique invoice URI
debts[] memo	Object	<i>Optional</i> Additional information related to the debit
debts[] authorized	Boolean	<i>Optional</i> Indicates whether the debit has been authorized by the required account holder
debts[] rejected	Boolean	<i>Optional</i> Indicates whether debit has been rejected by account holder
debts[] rejection_message	String	<i>Optional</i> Reason the debit was rejected
credits	Array	Funds that come out of the transfer
credits[] account	URI	Account receiving the funds
credits[] amount	String	Amount as decimal
credits[] invoice	URI	<i>Optional</i> Unique invoice URI
credits[] memo	Object	<i>Optional</i> Additional information related to the credit
credits[] authorized	Boolean	<i>Optional</i> Indicates whether the credit has been authorized by the required account holder
credits[] rejected	Boolean	<i>Optional</i> Indicates whether credit has been rejected by account holder
credits[] rejection_message	String	<i>Optional</i> Reason the credit was rejected
execution_condition	String	The condition for executing the transfer
expires_at	DateTime	Time when the transfer expires. If the transfer has not executed by this time, the transfer is canceled.
state	String	<i>Optional, Read-only</i> The current state of the transfer (informational only)
timeline	Object	<i>Optional, Read-only</i> Timeline of the transfer's state transitions
timeline.prepared_at	DateTime	<i>Optional</i> An informational field added by the ledger to indicate when the transfer was originally prepared
timeline.executed_at	DateTime	<i>Optional</i> An informational field added by the ledger to indicate when the transfer was originally executed

Account Object

An account represents a DFSP's position at the central ledger.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. An account object can have the following fields:

Name	Type	Description
id	URI	<i>Read-only</i> Resource identifier
name	String	Unique name of the account
password	String	Password for the account
balance	String	<i>Optional, Read-only</i> Balance as decimal
is_disabled	Boolean	<i>Optional, Read-only</i> Admin users may disable/enable an account
ledger	URI	<i>Optional, Read-only</i> A link to the account's ledger
created	DateTime	<i>Optional, Read-only</i> Time when account was created

User Object

An user represents an admin for the central ledger.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. An user object can have the following fields:

Name	Type	Description
userId	URI	<i>Read-only</i> User identifier
firstName	String	First name of the account
lastName	String	Last name for the account
email	String	Email as a string
isActive	Boolean	<i>Optional</i> Users may be disabled/enabled
createdDate	DateTime	<i>Optional, Read-only</i> Time when account was created

Notification Object

The central ledger pushes a notification object to WebSocket clients when a transfer changes state. This notification is sent at most once for each state change.

A notification object can have the following fields:

Name	Type	Description
resource	Object	<u>Transfer object</u> that is the subject of the notification

Name	Type	Description
related_resources	Object	<i>Optional</i> Additional resources relevant to the event
related_resources.execution_condition_fulfillment	String	<i>Optional</i> Proof of condition completion
related_resources.cancellation_condition_fulfillment	String	<i>Optional</i> Proof of condition completion

Metadata Object

The central ledger will return a metadata object about itself allowing client's to configure themselves properly.

A metadata object can have the following fields:

Name	Type	Description
currency_code	String	Three-letter (ISO 4217) code of the currency this ledger tracks
currency_symbol	String	Currency symbol to use in user interfaces for the currency represented in this ledger. For example, "\$"
ledger	URI	The ledger that generated the metadata
urls	Object	Paths to other methods exposed by this ledger. Each field name is short name for a method and the value is the path to that method.
precision	Integer	How many total decimal digits of precision this ledger uses to represent currency amounts
scale	Integer	How many digits after the decimal place this ledger supports in currency amounts

Position Object

The central ledger can report the current positions for all registered accounts.

A position object can have the following fields:

Name	Type	Description
account	URI	A link to the account for the calculated position
payments	String	Total non-settled amount the account has paid as string
receipts	String	Total non-settled amount the account has received as string
net	String	Net non-settled amount for the account as string

Charge Object

A charge represents a fee that will be applied on a transfer at execution

A charge object can have the following fields:

Name	Type	Description
id	Integer	Identifier for the charge
name	String	Unique name of the charge
charge_type	String	The type of the charge, should be <i>fee</i>
rate_type	String	The rate type for the charge, either <i>flat</i> or <i>percent</i>
rate	String	The amount for the charge, represented as a <i>decimal for the percent rate type</i> (<i>5%</i> is <i>0.05</i>) and as the <i>actual value for the flat rate type</i>
minimum	String	Minimum transfer amount for the charge to be applied
maximum	String	Maximum transfer amount for the charge to be applied
code	String	The letter code used to identify the charge
is_active	Boolean	Admin users may activate or deactivate the charge
payer	String	The account that pays the fee generated by the charge either <i>sender</i> , <i>receiver</i> , or <i>ledger</i>
payee	String	The account that receives the fee generated by the charge either <i>sender</i> , <i>receiver</i> , or <i>ledger</i>

Permission Object

The central-ledger uses permissions to manage the capabilities of an admin.

A permission object has the following fields:

Name	Type	Description
key	String	The key for the permission
description	String	A description of the permission

Role Object

The central-ledger uses roles to manage sets of permissions of an admin.

A role object has the following fields:

Name	Type	Description
roleId	String	Identifier for the role
name	String	Name of the role
description	String	Description of the role
permissions	Array	A list of permission keys

Error Information

This section identifies the potential errors returned and the structure of the response.

An error object can have the following fields:

Name	Type	Description
id	String	An identifier for the type of error
message	String	A message describing the error that occurred
validationErrors	Array	<i>Optional</i> An array of validation errors
validationErrors[].message	String	A message describing the validation error
validationErrors[].params	Object	An object containing the field that caused the validation error
validationErrors[].params.key	String	The name of the field that caused the validation error
validationErrors[].params.value	String	The value that caused the validation error
validationErrors[].params.child	String	The name of the child field

HTTP/1.1 404 Not Found

Content-Type: application/json

```
{  
  "id": "InvalidUriParameterError",  
  "message": "Error validating one or more uri parameters",  
  "validationErrors": [  
    {  
      "message": "id must be a valid GUID",  
      "params": {  
        "value": "7d4f2a70-e0d6-42dc-9efb-6d23060ccd6",  
        "key": "id"  
      }  
    }  
  ]  
}
```

```
    }  
}  
]  
}
```

/

Working with Transfers

Introduction

In this guide, we'll walk through the different steps of successfully executing a transfer: [Creating accounts](#) [Preparing a transfer](#) [Executing a transfer](#) [Next Steps](#)

Step 1: Creating accounts

To get started, you'll need to create two accounts, one to credit to and one to debit from.

Create account dfsp1

Start off by creating an account with the name **dfsp1**. Simply provide the account's name and password then make a call to the create account endpoint. More detail about the response and errors can be found in the [API documentation](#).

Request

```
POST http://central-ledger/accounts
Content-Type: application/json
{
  "name": "dfsp1",
  "password": "dfsp1_password"
}
```

Response

```
HTTP/1.1 201 Created
```

```
{  
  "id": "http://central-ledger/accounts/dfsp1",  
  "name": "dfsp1",  
  "created": "2017-01-03T22:29:46.068Z",  
  "balance": "0",  
  "is_disabled": false,  
  "ledger": "http://central-ledger"  
}
```

Create account dfsp2

Next, create an account with the name **dfsp2**. Like before, provide the account's name and password then make a call to the create account endpoint.

Request

```
POST http://central-ledger/accounts HTTP/1.1  
Content-Type: application/json  
{  
  "name": "dfsp2",  
  "password": "dfsp2_password"  
}
```

Response

```
HTTP/1.1 201 Created  
{  
  "id": "http://central-ledger/accounts/dfsp1",  
  "name": "dfsp2",  
  "created": "2017-01-03T22:30:46.068Z",  
  "balance": "0",  
  "is_disabled": false,  
  "ledger": "http://central-ledger"  
}
```

Step 2: Preparing a transfer

Now that you have two accounts created, a transfer can be prepared. For this transfer, you will be debiting 100 from

dfsp1 and crediting it to **dfsp2**. There are quite a few optional fields that can be passed on a transfer prepare call. For now, you can stick to the minimum. A more in depth explanation can be found in the [API documentation](#).

Request

```
PUT http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204 HTTP/1.1
Content-Type: application/json
{
  "id": "http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204",
  "ledger": "http://central-ledger",
  "debits": [
    {
      "account": "http://central-ledger/accounts/dfsp1",
      "amount": "100"
    }
  ],
  "credits": [
    {
      "account": "http://central-ledger/accounts/dfsp2",
      "amount": "100"
    }
  ],
  "execution_condition": "ni:///sha-256;47DEQpj8HBSa-TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
  "expires_at": "2017-12-31T00:00:01.000Z"
}
```

Response

```
HTTP/1.1 201 Created
{
  "id": "http://central-ledger/transfers/3a2a1d9e-8640-4d2d-b06c-84f2cd613204",
  "ledger": "http://central-ledger",
  "debits": [
    {
      "account": "http://central-ledger/accounts/dfsp1",
      "amount": 50
    }
  ],
  "credits": [
    {
      "account": "http://central-ledger/accounts/dfsp2",
      "amount": 50
    }
  ],
}
```

```
"execution_condition": "ni:///sha-256;47DEQpj8HBSa-  
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",  
  "expires_at": "2017-12-31T00:00:01.000Z",  
  "state": "prepared",  
  "timeline": {  
    "prepared_at": "2017-01-01T22:43:41.385Z"  
  }  
}
```

Step 3: Executing a transfer

Now that the transfer is prepared, you are free to execute the transfer. This consists of either fulfilling or cancelling the transfer. For this example, you will be fulfilling the transfer, because the transfer was prepared with an `execution_condition` as opposed to a `cancellation_condition`. Like before, more information can be found in the [API documentation](#).

Request

```
PUT http://central-ledger/transfers/3a2a1d9e-8640-4d2d-  
b06c-84f2cd613204/fulfilment HTTP/1.1  
Content-Type: text/plain  
oAKAAA
```

Response

```
HTTP/1.1 200 OK  
oAKAAA
```

Next Steps

Now that you have successfully created accounts and prepared and executed a transfer, you should feel comfortable working with the other endpoints found in the [API documentation](#).

central-ledger

The central ledger is a series of services that facilitate clearing and settlement of transfers between DFSPs, including the following functions:

- Brokering real-time messaging for funds clearing
- Maintaining net positions for a deferred net settlement
- Propagating scheme-level and off-transfer fees

The following documentation represents the services, APIs and endpoints responsible for various ledger functions.

Contents:

- [Deployment](#)
- [Configuration](#)
- [API](#)
- [Logging](#)
- [Tests](#)

Deployment

See the [Onboarding guide](#) for running the service locally.

Configuration

Environment variables

The Central Ledger has many options that can be configured through environment variables.

Environment variable	Description	Example values
CLEDG_DATABASE_URI	The connection string for the database the central ledger will use. Postgres is currently the only supported database.	postgres://:@localhost:5432/central_ledger
CLEDG_PORT	The port the API server will run on.	3000
CLEDG_ADMIN_PORT	The port the Admin server will run on.	3001
CLEDG_HOSTNAME	The URI that will be used to create and validate links to resources on the central ledger.	http://central-ledger
CLEDG_ENABLE_BASIC_AUTH	Flag to enable basic auth protection on endpoints that require authorization. Username and password would be the account name and password.	false
CLEDG_ENABLE_TOKEN_AUTH	Flag to enable token protection on endpoints that require authorization. To create a token, reference the API documentation .	false
CLEDG_LEDGER_ACCOUNT_NAME	Name of the account setup to receive fees owed to the central ledger. If the account doesn't exist, it will be created on start up.	LedgerName
CLEDG_LEDGER_ACCOUNT_PASSWORD	Password of the account setup to receive fees owed to the central ledger.	LedgerPassword
CLEDG_ADMIN_KEY	Key used for admin access to endpoints that require validation.	AdminKey
CLEDG_ADMIN_SECRET	Secret used for admin access to endpoints that require validation. Secret also used to sign JWTs used for Admin API.	AdminSecret
CLEDG_TOKEN_EXPIRATION	Time in milliseconds for Admin API tokens to expire.	3600000

Environment variable	Description	Example values
CLEDG_EXPIRES_TIMEOUT	Time in milliseconds to determine how often transfer expiration process runs.	5000
CLEDG_AMOUNT_PRECISION	Numeric value used to determine precision recorded for transfer amounts on this ledger.	10
CLEDG_AMOUNT_SCALE	Numeric value used to determine scale recorded for transfer amounts on this ledger.	2

API

For endpoint documentation, see the [API documentation](#).

For help preparing and executing transfers, see the [Transfer Guide](#)

Logging

Logs are sent to standard output by default.

Tests

Tests include unit, functional, and integration.

Running the tests:

```
npm run test:all
```

Tests include code coverage via istanbul. See the test/ folder for testing scripts.

Central Ledger Setup

Introduction

In this document we'll walk through the setup for the Mojaloop Central Ledger. It consists of three sections:

- [Software List](#)
 - [Setup](#)
 - [Errors On Setup](#)
-

Software List

1. Github
 2. brew
 3. Docker
 4. MySQLWorkbench
 5. Postman
 6. nvm
 7. npm
 8. Zenhub
 9. central_ledger
 10. JavaScript IDE
-

Setup

Make sure you have access to [Mojaloop on Github](#) and clone the project.

Installing brew

macOS

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Ubuntu

To install Linuxbrew, follow these [instructions](#)

Installing Docker

To install Docker, follow these instructions: [Docker for Mac](#), [Docker for Ubuntu](#)

Installing MySQL

Docker

Run the following commands in your terminal. Please ensure that you run the MySQL statements with the semicolon at the end.

```
DBUSER=central_ledger; DBPASS=password; SLEEPTIME=15; docker stop mysql; docker rm mysql; docker run -p 3306:3306 -d --name mysql -e MYSQL_USER=$DBUSER -e MYSQL_PASSWORD=$DBPASS -e MYSQL_DATABASE=$DBUSER -e MYSQL_ALLOW_EMPTY_PASSWORD=true mysql/mysql-server; sleep $SLEEPTIME; docker exec -it mysql mysql -uroot -e "ALTER USER '$DBUSER'@'%' IDENTIFIED WITH mysql_native_password BY '$DBPASS';"
```

Installing MySQLWorkBench

macOS

Go to and follow the instructions
<https://dev.mysql.com/downloads/workbench/>

Ubuntu

For pgAdmin 4 v2.1 on Ubuntu, according to the [download page](#):

Install dependencies, create a virtual environment, download, install & configure:

```
sudo apt-get install virtualenv python-pip libpq-dev python-dev  
cd  
virtualenv pgadmin4  
cd pgadmin4  
source bin/activate  
  
pip install https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v2.1/pip/  
pgadmin4-2.1-py2.py3-none-any.whl
```

Override default paths and set it to single-user mode in the [local configuration file](#):

```
nano lib/python2.7/site-packages/pgadmin4/config_local.py
```

Write:

```
import os  
DATA_DIR = os.path.realpath(os.path.expanduser(u'~/.pgadmin/'))  
LOG_FILE = os.path.join(DATA_DIR, 'pgadmin4.log')  
SQLITE_PATH = os.path.join(DATA_DIR, 'pgadmin4.db')  
SESSION_DB_PATH = os.path.join(DATA_DIR, 'sessions')  
STORAGE_DIR = os.path.join(DATA_DIR, 'storage')  
SERVER_MODE = False
```

Make a shortcut:

```
touch ~/pgadmin4/start  
chmod +x ~/pgadmin4/start  
nano ~/pgadmin4/start
```

Write:

```
#!/bin/bash  
cd ~/pgadmin4  
source bin/activate  
python lib/python2.7/site-packages/pgadmin4/pgAdmin4.py
```

Run with `~/pgadmin4/start` and access at <http://localhost:5050>

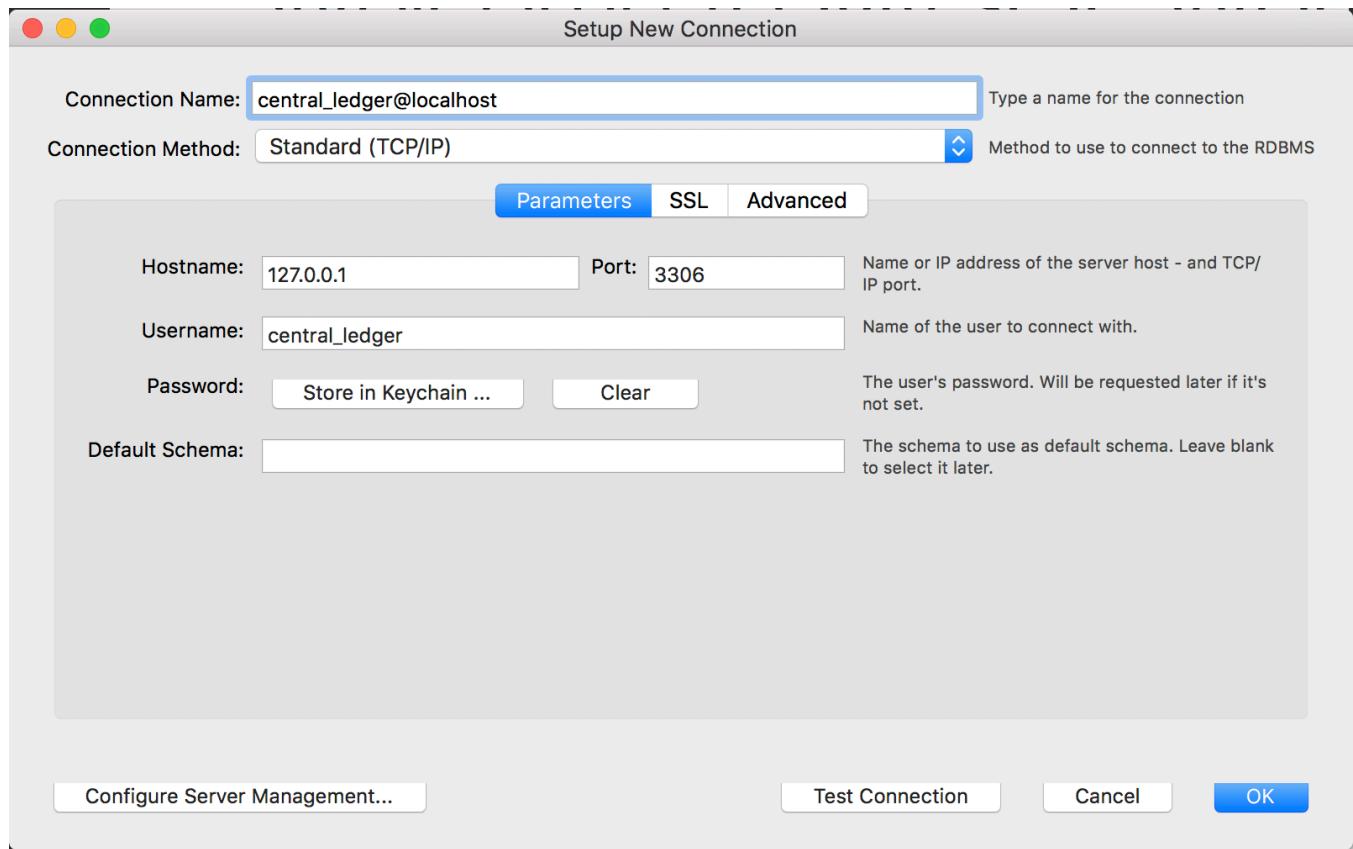
Setup MySQLWorkbench

Please follow the below instructions:

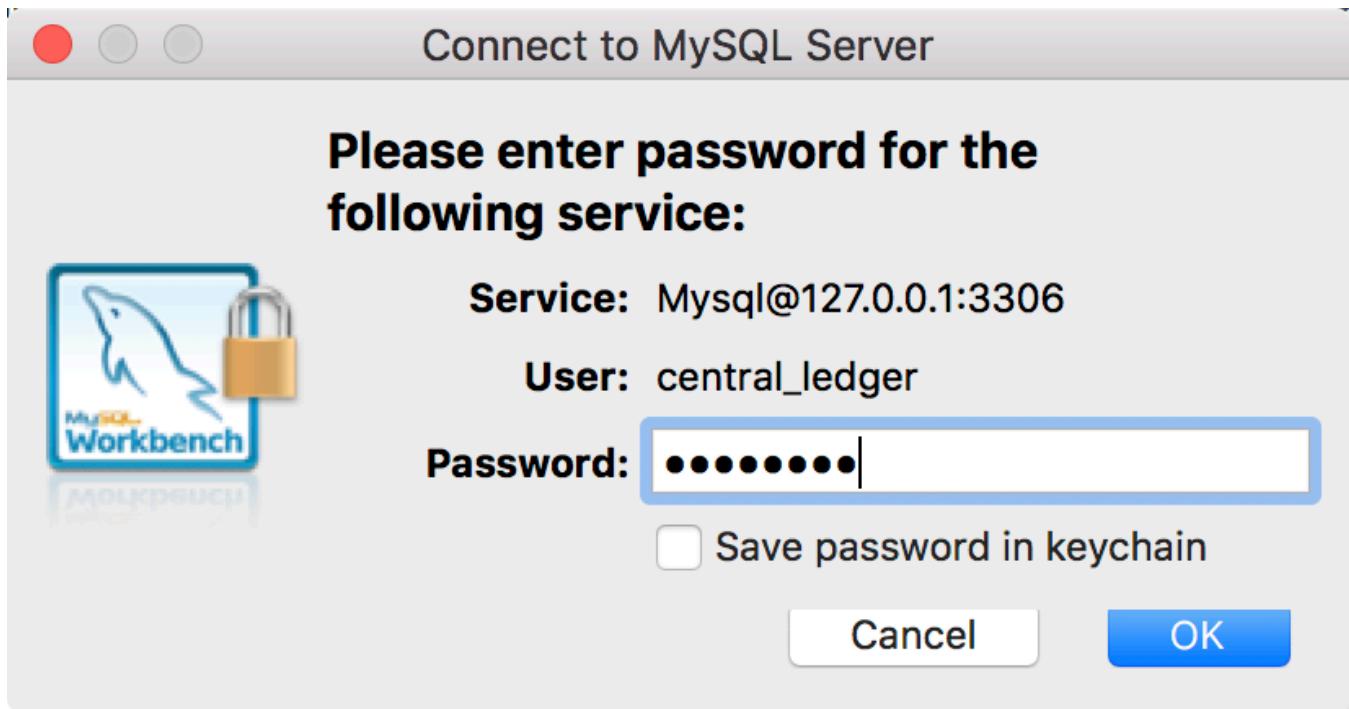
- Click the add (+) icon

MySQL Connections

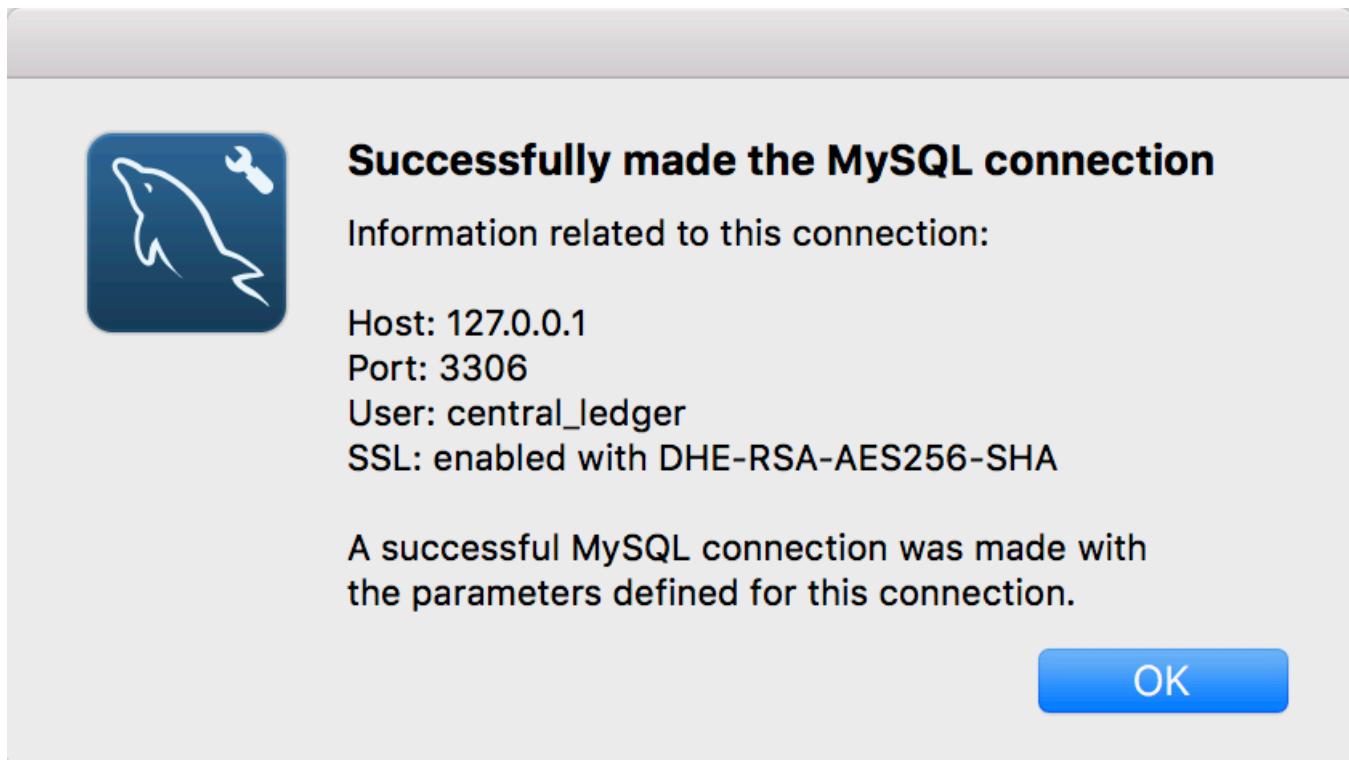
- Enter the Connection name and username as per image and click test connection



- Enter the password => 'password' click OK

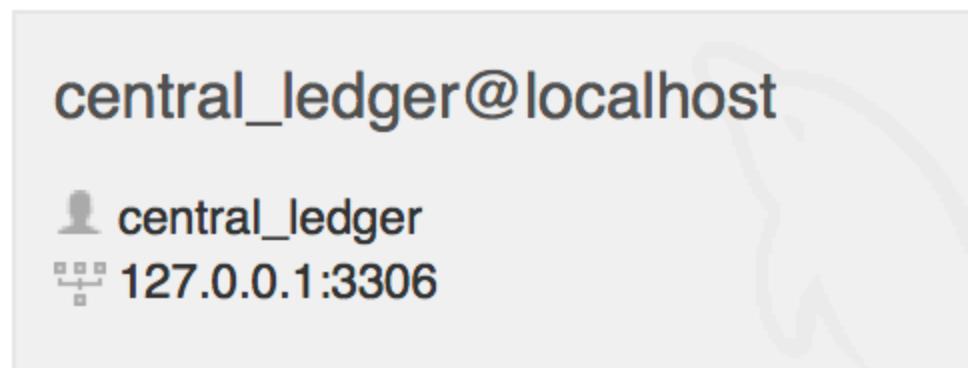


d. You should see this click OK

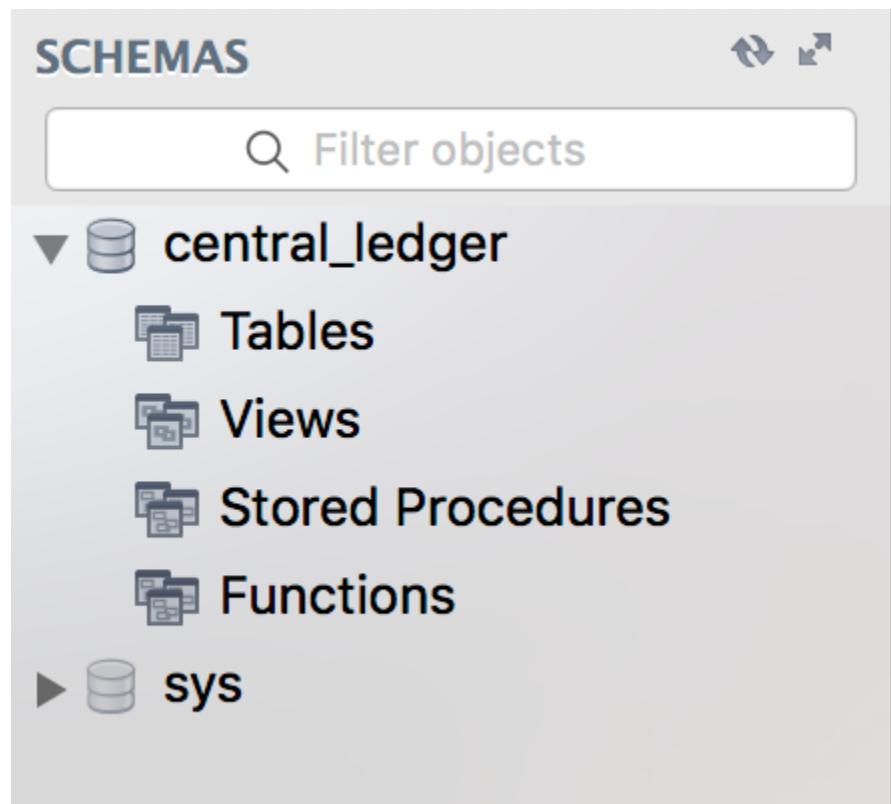


e. This should now be shown on you MySQLWorkbench dashboard

MySQL Connections + ✖



f. You should see the central_ledger database under schema no tables will be present but will get populated when you start your server



Installing Postman

Please, follow these instructions: [Get Postman](#)

Alternatively on **Ubuntu** you may run:

```
wget https://dl.pstmn.io/download/latest/linux64 -O postman.tar.gz  
sudo tar -xzf postman.tar.gz -C /opt  
rm postman.tar.gz  
sudo ln -s /opt/Postman/Postman /usr/bin/postman
```

Setup Postman

- open *Postman*
- click **Import** and then **Import File**
- navigate to the central_ledger directory and select (to be added) and import it into postman

nvm

(This is optional, you can install node directly from the website, node version manager(nvm) isn't really needed unless you want to use multiple versions of node)

If you don't have cURL already installed, on **Ubuntu** run `sudo apt install curl`

Download the nvm install via Homebrew:

```
brew update  
brew install nvm  
mkdir ~/.nvm  
vi ~/.bash_profile
```

- Ensure that nvm was installed correctly with `nvm --version`, which should return the version of nvm installed
- Install the version (at time of publish 8.9.4 current LTS) of Node.js you want:
 - Install the latest LTS version with `nvm install --lts`
 - Use the latest LTS verison with `nvm use --lts`
 - Install the latest version with `nvm install node`
 - Use the latest version with `nvm use node`

- If necessary, fallback to nvm install 8.9.4

Setup nvm

Create a `.bash_profile` file with `touch ~/.bash_profile`, then `nano ~/.bash_profile` and *write*:

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh" # This loads nvm
```

npm

By installing `node` during `nvm` installation above, you should have the corresponding npm version installed

Setup npm

- The `.npmrc` file in your user root just needs to be present as the repository it will use is `http://npmjs.org` If it doesn't exist just create it.

Installing ZenHub for GitHub

Open Google Chrome browser and navigate to [Zenhub Extension](#)

Installing central_ledger

- `cd` into the `central_ledger` project and run subsequently the following commands:

```
npm install -g node-gyp (needs to be done once)
brew install libtool autoconf automake (needs to be done once)
npm install
source ~/.bash_profile
npm rebuild
```

- set `CLEDG_DATABASE_URI` environment variable:

```
export
CLEDG_DATABASE_URI=mysql://central_ledger:password@localhost:3306/
central_ledger
```

- disable SIDECAR in **config/default.json** temporary by setting "SIDECAR": {
 "DISABLED": "true", ...}
- run npm start (*to run it locally*) or npm run dev (*to run it on your Docker host*)

Run Postman

- click on **Central Ledger** and then **Prepare transfer**
- click **Send**
- if you get a valid response, you should be ready to go

Errors On Setup

- ./src/argon2_node.cpp:6:10: fatal error: 'tuple' file not found
- resolved by running CXX='clang++ -std=c++11 -stdlib=libc++' npm rebuild
- sodium v1.2.3 can't compile during npm install
- resolved by installing v2.0.3 npm install sodium@2.0.3

Running central-ledger on Kubernetes

1. Install VirtualBox Refer to: <https://www.virtualbox.org/wiki/Downloads>
2. Install Docker MacOS: `brew install docker`
3. Install Kubectl MacOS: `brew install kubectl`
4. Install Minikube MacOS: `brew cask install minikube`
5. Install Helm MacOS: `brew install kubernetes-helm`
6. Initialise MiniKube `minikube start`
7. Initialise Helm `helm init <-- this only needs to be done once`
8. Deploy Ingress `minikube addon enable ingress`
9. Configure PostgreSQL Edit `postgresUser` & `postgresPassword` as desired in the following file
`./deploy/helm/central-ledger-helm-postgresql-values.yaml`
10. Deploy PostgreSQL `helm install --name central-ledger -f ./deploy/helm/central-ledger-helm-postgresql-values.yaml stable/postgresql`
11. Configure credentials in the Central-ledger-secret Edit `db.uri` with the details from step 10 above in the following file `./deploy/k8s/central-ledger-secret.yaml`.

Ensure the values are base64 encoded.

12. Deploy Central-ledger `kubectl create -f ./deploy/k8s`

Or alternatively you can stipulate a namespace for deployment `kubectl -n dev create -f ./deploy/k8s`

13. Add the following to your hosts file `<IP> central-ledger.local`

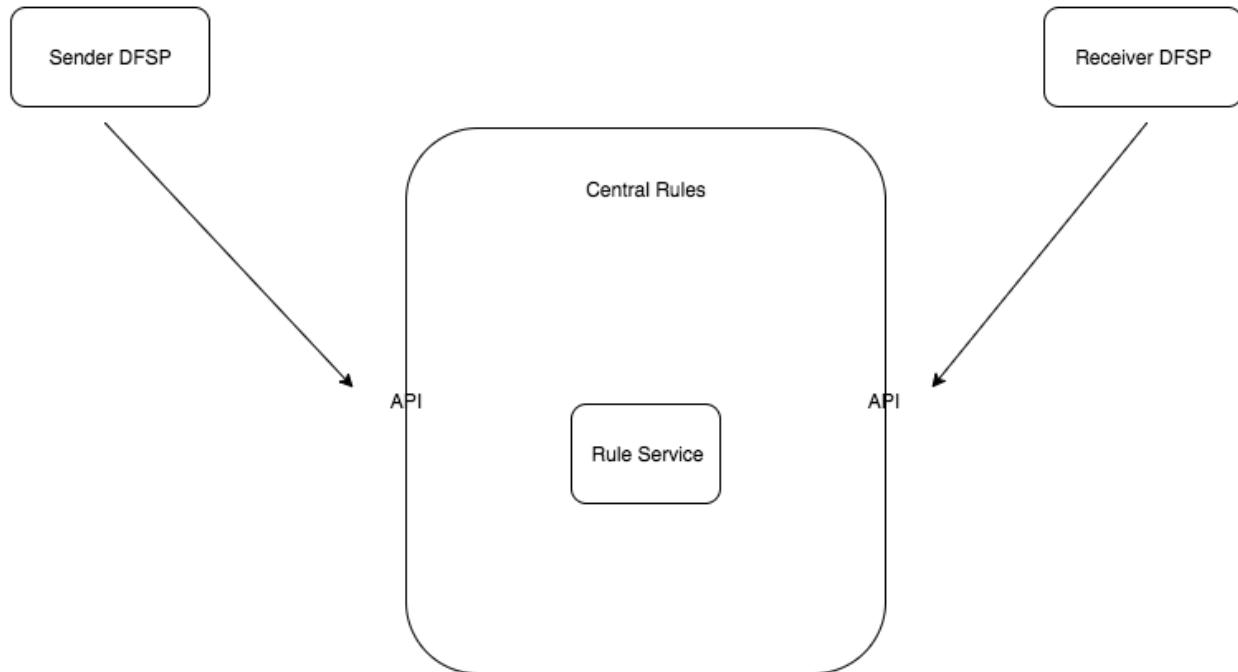
Where `<IP>` can be attained using the following command `minikube ip`

14. Open K8s Dashboard

```
minikube dashboard
```

Central Rules

Component Architecture



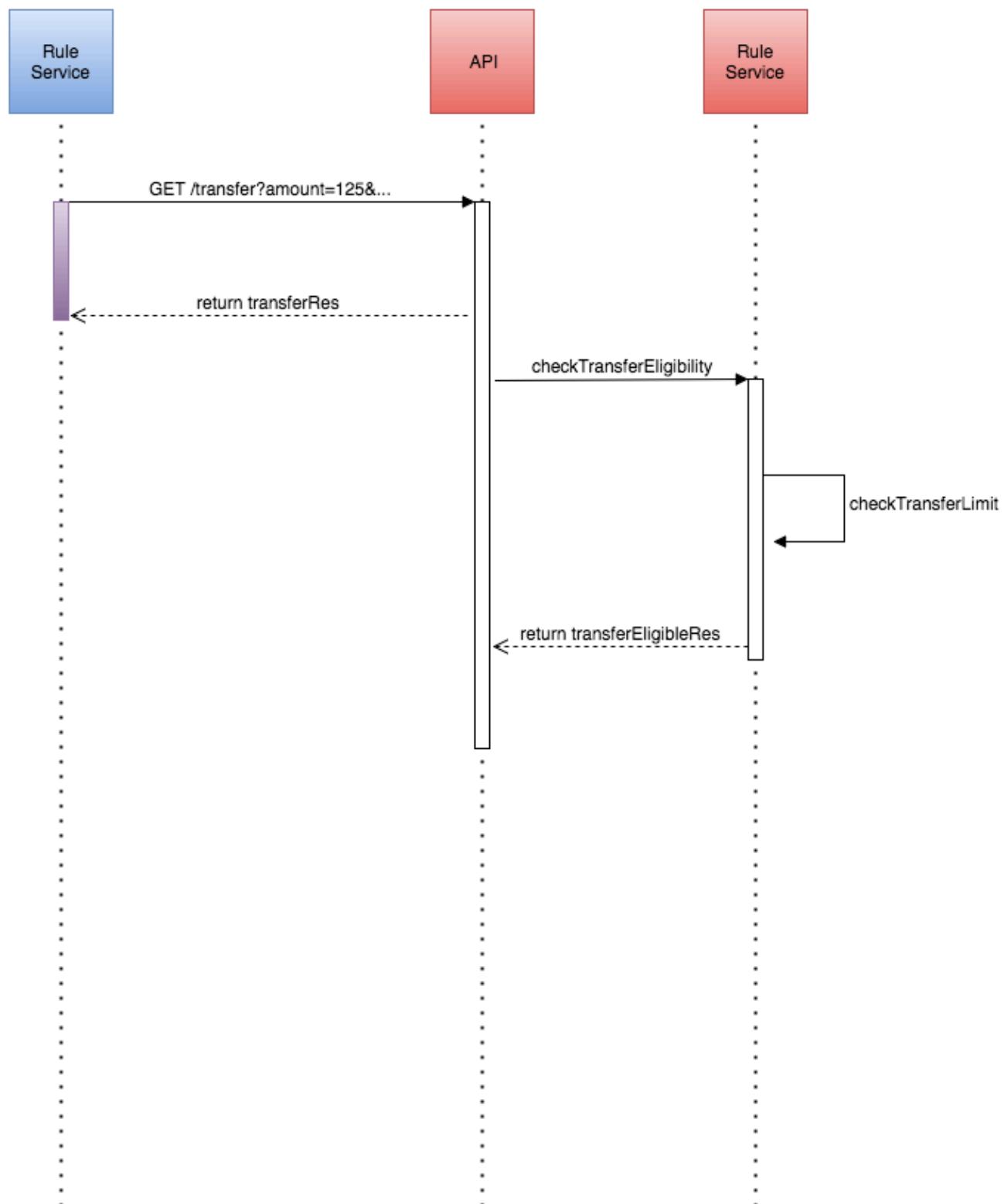
Component	Function	Interaction
API	Handles calls for checking transfer eligibility.	Calls internal service to determine if transfer is eligible.
Rule Service	Handles checking and validating transfers against any rules enforced by the scheme.	Called by API.

Assumptions (for 0.5)
<ul style="list-style-type: none"> DSFPs will pass in all transfer information DSFPs will also maintain their own transfer rules The Central Rules engine will: <ul style="list-style-type: none"> Focus on hub-wide overarching rules Will provide a success indicator and message Focus on a boolean success/failure, with the ability to expand to warnings Current scope focused on a transfer amount limit (e.g. no transfer greater than \$100,000)

Check Transfer Eligibility Flow

DFSP

Central Rules



Endpoints

[Endpoint documentation](#)

Central Rules Endpoints

The Central Rules API determines whether a transfer to an End User is permitted.

Endpoints

Check transfer eligibility

This endpoint is used to find out if a transfer to an End User is permitted.

Allowed

```
GET https://central-rules/  
transfer?sender_user_number=11144455555555&receiver_user_number=11122  
23333333&amount=125
```

```
HTTP/1.1 200 OK  
Content-Type: application/json  
{  
  "allowed": true  
}
```

Not allowed

```
GET https://central-rules/  
transfer?sender_user_number=11144455555555&receiver_user_number=11122  
23333333&amount=100001
```

```
``` http HTTP/1.1 200 OK Content-Type: application/json { "allowed": false, "reason": { "code":  
"transfer_limit_exceeded", "message": "Not allowed to send more than 100,000" } }
```

# Central Fraud Sharing API Documentation

---

In this guide, we'll walk through the different central fraud sharing endpoints: `POST User fraud score` `POST Transfer fraud score`

The different endpoints often deal with these [data structures](#): `Score Object` `Transfer Object`

---

## Endpoints

### User fraud score

This endpoint returns the fraud score for a given user

#### HTTP Request

```
POST http://central-fraud-sharing/score/user
```

#### Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

## Request body

Field	Type	Description
identifier	String	The identifier type and identifier to be created, separated by a colon

## Response 200 OK

Field	Type	Description
Object	Score	The <a href="#">Score object</a> generated

## Request

```
POST http://central-fraud-sharing/score/user HTTP/1.1
Content-Type: application/json
{
 "identifier": "tel:1112223456",
}
```

## Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
 "score": "48",
 "created": "2017-01-03T16:16:18.958Z",
 "id": "7d4f2a70-e0d6-42dc-9efb-6d23060cccd6",
}
```

## Testing

To test specifically for high fraud or low fraud users, include one of the listed Ids somewhere in the tel or eur.

Id	Type	Example	Returned
111	Low Fraud	123-456-0111	1
999	High Fraud	123-456-0999	99
100	Blacklisted	123-456-0100	100

## Transfer fraud score

This endpoint returns the fraud score for a given transfer

### HTTP Request

```
POST http://central-fraud-sharing/score/transfer
```

### Headers

Field	Type	Description
Content-Type	String	Must be set to application/json

### Request body

Field	Type	Description
Object	Transfer	The transfer on which the fraud score will be calculated

### Response 200 OK

Field	Type	Description
Object	Score	The <a href="#">Score object</a> generated

#### Request

```
POST http://central-ledger/transfers/
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f HTTP/1.1
Content-Type: application/json
{
 "id": "http://central-ledger/transfers/
2d4f2a70-e0d6-42dc-9efb-6d23060ccd6f",
 "ledger": "http://central-ledger",
 "debits": [
 {"account": "http://central-ledger/accounts/dfspl",
 "amount": "50"
]
},
```

```

"credits": [
 "account": "http://central-ledger/accounts/dfsp2",
 "amount": "50"
],
"execution_condition": "ni:///sha-256;47DEQpj8HBSa-
_TImW-5JCeuQeRkm5NMpJWZG3hSuFU?fpt=preimage-sha-256&cost=0",
"expires_at": "2016-12-26T00:00:01.000Z"
}

```

## Response

```

HTTP/1.1 200 OK
Content-Type: application/json
{
 "score": "82",
 "created": "2017-01-03T16:16:18.958Z",
 "guid": "6d4g2a82-e0f6-41dc-8efb-6d27060cccd6",
}

```

## Testing

To test specifically for high fraud or low fraud transfer, include one of the listed Ids somewhere in either account uri. If more than one Id is included, it will return the higher of the fraud scores. If none of the scenarios below are leveraged, a score randomly assigned from 1-99 will be returned.

Id	Type	Example	Returned
111	Low Fraud	http://central-ledger/accounts/dfsp111	1
999	High Fraud	http://central-ledger/accounts/dfsp999	99
100	Blacklisted	http://central-ledger/accounts/dfsp100	100

# Data Structures

## Transfer Object

A transfer represents money being moved between two DFSP accounts at the central ledger.

The transfer must specify an execution\_condition, in which case it executes automatically when presented with the fulfillment for the condition. (Assuming the transfer has not expired or been canceled first.) Currently, the central ledger only supports the condition type of PREIMAGE-SHA-256 and a max fulfillment length of 65535.

Some fields are Read-only, meaning they are set by the API and cannot be modified by clients. A transfer object can have the following fields:

Name	Type	Description
id	URI	Resource identifier
ledger	URI	The ledger where the transfer will take place
debts	Array	Funds that go into the transfer
debts[].account	URI	Account holding the funds
debts[].amount	String	Amount as decimal
debts[].invoice	URI	<i>Optional</i> Unique invoice URI
debts[].memo	Object	<i>Optional</i> Additional information related to the debit
debts[].authorized	Boolean	<i>Optional</i> Indicates whether the debit has been authorized by the required account holder
debts[].rejected	Boolean	<i>Optional</i> Indicates whether debit has been rejected by account holder
debts[].rejection_message	String	<i>Optional</i> Reason the debit was rejected
credits	Array	Funds that come out of the transfer
credits[].account	URI	Account receiving the funds
credits[].amount	String	Amount as decimal
credits[].invoice	URI	<i>Optional</i> Unique invoice URI
credits[].memo	Object	<i>Optional</i> Additional information related to the credit
credits[].authorized	Boolean	<i>Optional</i> Indicates whether the credit has been authorized by the required account holder
credits[].rejected	Boolean	<i>Optional</i> Indicates whether credit has been rejected by account holder
credits[].rejection_message	String	<i>Optional</i> Reason the credit was rejected
execution_condition	String	The condition for executing the transfer
expires_at	DateTime	Time when the transfer expires. If the transfer has not executed by this time, the transfer is canceled.
state	String	<i>Optional, Read-only</i> The current state of the transfer (informational only)
timeline	Object	<i>Optional, Read-only</i> Timeline of the transfer's state transitions

Name	Type	Description
timeline.prepared_at	DateTime	<i>Optional</i> An informational field added by the ledger to indicate when the transfer was originally prepared
timeline.executed_at	DateTime	<i>Optional</i> An informational field added by the ledger to indicate when the transfer was originally executed

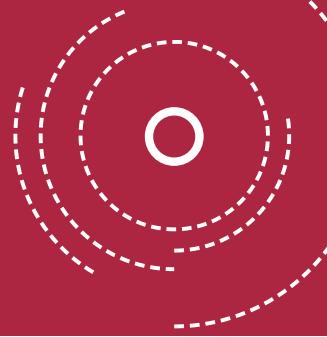
## Score Object

A score represents the likelihood of fraud for a given user or transaction, 0 being least likely, 100 being most likely.

A score object can have the following fields:

Name	Type	Description
score	Integer	The likelihood of fraud
created	DateTime	Time when score was created
id	URI	Resource identifier

# Infrastructure



## Logging Setup

Steps to perform ELK 5.X Stack Installation in AWS EC2 Instance RHEL

### JDK 8 Installation

=====

If not already installed JDK 8 must be installed to continue with the ELK 5.X Setup.

#### Install wget to download JDK 8 rpm

```
yum -y install wget
```

#### Download JDK 8 rpm

```
wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u121-b13/e9e7ea248e2c4826b92b3f075a80e441/jdk-8u121-linux-x64.rpm
```

#### Check JDK 8 rpm sha256 sum

```
sha256sum jdk-8u121-linux-x64.rpm
```

## Compare JDK 8 rpm sha256 sum against

<https://www.oracle.com/webfolder/s/digest/8u121checksum.html>

## Install JDK 8 rpm

```
rpm -ivh jdk-8u121-linux-x64.rpm
```

## Set Java default

```
java -version
```

If not 1.8.0\_121, make it your default java using the alternatives command:

```
sudo alternatives -config java
```

Enter the selection number to choose which java executable should be used by default.

# Elasticsearch Installation

---

## Import Elasticsearch PGP Key

```
rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
```

# Create file with elasticsearch repository information

```
vi /etc/yum.repos.d/elasticsearch.repo
```

Add following contents:

```
[elasticsearch-5.x]

name=Elasticsearch repository for 5.x packages

baseurl=https://artifacts.elastic.co/packages/5.x/yum

gpgcheck=1

gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch

enabled=1

autorefresh=1

type=rpm-md
```

## Install elasticsearch

```
yum -y install elasticsearch
```

## Configure elasticsearch

```
sudo vim /etc/elasticsearch/elasticsearch.yml
```

Go to Network section and modify the network.host:

```
network.host: [_eth0_, _local_]
```

## Start/Stop/Restart elasticsearch

```
sudo service elasticsearch start
```

```
sudo service elasticsearch stop
```

```
sudo service elasticsearch restart
```

## Make elasticsearch more verbose by removing the "-- quite \\" flag

```
vi /usr/lib/system/system/elasticsearch.service
```

## Restart elasticsearch service and perform daemon reload

```
sudo service elasticsearch restart
```

## Check that elasticsearch is running

Install netcat if not already install for debugging purposes:

```
yum -y install nmap-ncat
```

```
#ncat -v localhost 9200
```

```
Ncat: Version 6.40 (http://nmap.org/ncat)
```

```
Ncat: Connected to ::1:9200.
```

```
GET /
```

```
HTTP/1.0 404 Not Found
```

```
es.index_uuid: _na_
```

```
es.resource.type: index_or_alias
```

```
es.resource.id: bad-request
```

```
es.index: bad-request
```

```
content-type: application/json; charset=UTF-8
```

```
content-length: 367
```

```
{"error":{"root_cause":[{"type":"index_not_found_exception","reason":"no such index","resource.type":"index_or_alias","resource.id":"bad-request","index_uuid":"_na_","index":"bad-request"}],"type":"index_not_found_exception","reason":"no such index","resource.type":"index_or_alias","resource.id":"bad-request","index_uuid":"_na_","index":"bad-request"},"status":404}
```

If localhost is not the hostname, specify correct hostname or use server ip here. Also, must type "**GET \**".

Can also use curl command to check that elasticsearch is running:

```
curl -X GET http://localhost:9200/
```

# Kibana Installation

## Install kibana

```
yum -y install kibana
```

## Configure Kibana

```
vi /etc/kibana/kibana.yml
```

Uncomment and make sure to set the following 4 entries:

```
server.port: 5601
server.host: "localhost"
server.name: "localhost"
elasticsearch.url: http://localhost:9200
```

NOTE: localhost needs to be replaced by the actual hostname or server ip depending on ELK stack configuration, currently entire ELK stack is running on the same server.

## Start/Stop/Restart Kibana

```
sudo service kibana start
```

```
sudo service kibana stop
```

```
sudo service kibana restart
```

## Verify that you Kibana can be accessed from the browser

<http://localhost:5601/app/kibana>

NOTE: localhost needs to be replaced by the actual hostname or server ip. If no UI is available, go to NGINX reverse proxy section to access Kibana.

## Verify Kibana status from the browser

<http://localhost:5601/status>

NOTE: If no UI installed, continue to NGINX reverse proxy installation section to access Kibana from the browser.

# Logstash Installation

NOTE: Currently not installed as Beat log shippers (Filebeat and Metricsbeat) are directly sending logs to Elasticsearch. Logstash can be used to perform processing of logs. For more information, look at Additional Considerations section.

## Install Logstash

```
yum -y install logstash
```

## Start/Stop/Restart Logstash

```
sudo service kibana start
```

```
sudo service kibana stop
```

```
sudo service kibana restart
```

### Example Logstash Configuration to read system logs (/var/logs/\*log)

```
input {
 file {
 type => "syslog"
 path => ["/var/log/messages", "/var/log/*.log"]
 }
}

output {
```

```
stdout {
 codec => rubydebug
}

elasticsearch {

 host => "localhost" # Use the internal IP of your Elasticsearch server

 # for production

}

}
```

## BEST PRACTICES

-Separate large Logstash configuration files into several smaller ones. Conf file path can be set to a directory. Files in directory will be merged by name, therefore name logstash configuration files in alphabetical order.

-Configure Filebeat to feed Logstash and Logstash to feed Elasticsearch.

# Filebeat Installation

## Install Filebeat

```
yum -y install filebeat
```

## Configure Filebeat

```
vi /etc/filebeat/filebeat.yml
```

Configure Filebeat to ship files to Elasticsearch:

Under Elasticsearch output section modify

```
hosts: ["http://172.31.45.32:9200"]
```

NOTE: Use the eth0 ip where Elasticsearch is running.

Configure path for Filebeat to crawl and fetch logs from:

Under Filebeat prospectors section, identify paths and for example add

- /var/log/mule\_logs/mule\_dfsp1/\*.log

## Start/Stop/Restart Filebeat

```
sudo service filebeat start
```

```
sudo service filebeat stop
```

```
sudo service filebeat restart
```

## Import dashboards and index

```
/usr/share/filebeat/scripts/import_dashboards
```

## Logstash vs Beats

Beats are lightweight data shippers that you install as agents on your servers to send specific types of operational data to Elasticsearch. Beats have a small footprint and use fewer system resources than Logstash.

Logstash has a larger footprint, but provides a broad array of input, filter, and output plugins for collecting, enriching, and transforming data from a variety of sources.

Beats are lightweight data shippers that you install as agents on your servers to send specific types of operational data to Elasticsearch. Beats have a small footprint and use fewer system resources than Logstash.

Logstash has a larger footprint, but provides a broad array of input, filter, and output plugins for collecting, enriching, and transforming data from a variety of sources.

Among Logstash filters that can be leveraged at L1P are; anonymize, json.

# Metricbeat Installation

## Install Metricbeat

```
yum -y install metricbeat
```

## Configure Metricbeat

```
vi /etc/metricbeat/metricbeat.yml
```

Configure Metricbeat to ship files to Elasticsearch:

Under Elasticsearch output section modify

```
hosts: ["http://172.31.45.32:9200"]
```

NOTE: Use the eth0 ip where Elasticsearch is running.

## Start/Stop/Restart Metricbeat

```
sudo service metricbeat start
```

```
sudo service metricbeat stop
```

```
sudo service metricbeat restart
```

## Import dashboards and index

```
/usr/share/metricbeat/scripts/import_dashboards
```

# NGINX Reverse Proxy Installation

## Install NGINX

```
yum -y install nginx httpd-tools
```

## Create password file for basic authentication of http users

```
htpasswd -c /etc/nginx/conf.d/kibana.hpasswd admin
```

## Configure NGINX

```
vi /etc/nginx/conf.d/kibana.conf

server {
 listen 80;
 server_name localhost;
 auth_basic "Restricted Access";
 auth_basic_user_file /etc/nginx/htpasswd.users;

 location / {
 proxy_pass http://localhost:5601;
 proxy_http_version 1.1;
 proxy_set_header Upgrade \$http_upgrade;
 proxy_set_header Connection 'upgrade';
 proxy_set_header Host \$host;
 proxy_cache_bypass \$http_upgrade;
 }
}
```

## Restart NGINX

```
sudo service kibana restart
```

## Access Kibana via NGINX on your browser

[http://EC2\\_INSTANCE\\_URL](http://EC2_INSTANCE_URL)

Enter username/password admin/adminpassword

## Modify AWS EC2 Instance Security Group to open ports

### Create Two Inbound rules

1. tcp for port 80 for NGINX
2. tcp for port 9200 for Elasticsearch, can be opened only for Beats/Logstash servers

## Kibana Query

<https://www.elastic.co/guide/en/kibana/current/search.html>

<https://www.mjt.me.uk/posts/kibana-101/>

<https://www.timroes.de/2016/05/29/elasticsearch-kibana-queries-in-depth-tutorial/>

<http://logz.io/blog/kibana-tutorial/>

To perform Kibana Queries log into kibana. Make sure to set proper Time Range at Kibana->Discover on top right hand corner. Simply enter query and search.

# Custom L1P\_Index Configuration

The custom L1P\_Index is defined by the two files show below (ilp\_template.json and types.json) which are part of the interop-elk GitHub repo. The L1P\_Index is an elasticsearch index used as the storage data structure for the L1P specific log data. The main purpose is to capture and display L1P transaction timestamp across L1P components.

ilp\_template.json:

```
{
 "template": [
 "l1p_index*"
],
 "mappings": {
 "l1p_log": {
 "_all": {
 "norms": false
 },
 "dynamic_templates": [
 {
 "strings_as_keyword": {
 "match_mapping_type": "string",
 "mapping": {
 "ignore_above": 1024,
 "type": "keyword"
 }
 }
 }
 }
 }
 }
}
```

```
],
 "properties": {
 "@timestamp": {
 "type": "date"
 },
 "l1p_trace_id": {
 "type": "keyword"
 },
 "beat": {
 "properties": {
 "hostname": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "name": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "version": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "processing_timestamp": {
 "type": "date"
 }
 }
 }
 }
}
```

```
},
 "input_type": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "message": {
 "type": "text",
 "norms": false
 },
 "meta": {
 "properties": {
 "cloud": {
 "properties": {
 "availability_zone": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "instance_id": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "machine_type": {
 "type": "keyword",
 "ignore_above": 1024
 },
 "project_id": {
 "type": "keyword",
 "ignore_above": 1024
 }
 }
 }
 }
 }
}
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"provider": {
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"region": {
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
}
```

```
}
```

```
}
```

```
},
```

```
"offset": {
```

```
"type": "long"
```

```
},
```

```
"source": {
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"tags": {
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
 }
```

```
}
```

```
}
```

```
}
```

types.json

```
{
```

```
 "l1p_index": {
```

```
 "mappings": {
```

```
 "l1p_log": {
```

```
 "_all": {
```

```
 "norms": false
```

```
 },
```

```
 "dynamic_templates": [
```

```
 {
```

```
 "strings_as_keyword": {
```

```
 "match_mapping_type": "string",
```

```
 "mapping": {
```

```
 "ignore_above": 1024,
```

```
 "type": "keyword"
```

```
 }
```

```
 }
```

```
 }
```

```
],
```

```
 "properties": {
```

```
"@timestamp": {
 "type": "date"
},

"ilp_trace_id": {
 "type": "keyword"
},

"beat": {
 "properties": {
 "hostname": {
 "type": "keyword",
 "ignore_above": 1024
 },

 "name": {
 "type": "keyword",
 "ignore_above": 1024
 },

 "version": {
 "type": "keyword",
 "ignore_above": 1024
 },

 "processing_timestamp": {
 "type": "date"
 }
 }
},

"input_type": {
```

```
"type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"message": {
```

```
 "type": "text",
```

```
 "norms": false
```

```
},
```

```
"meta": {
```

```
 "properties": {
```

```
 "cloud": {
```

```
 "properties": {
```

```
 "availability_zone": {
```

```
 "type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"instance_id": {
```

```
 "type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"machine_type": {
```

```
 "type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"project_id": {
```

```
 "type": "keyword",
```

```
"ignore_above": 1024
```

```
},
```

```
"provider": {
```

```
 "type": "keyword",
```

```
 "ignore_above": 1024
```

```
},
```

```
"region": {
```

```
 "type": "keyword",
```

```
 "ignore_above": 1024
```

```
}
```

```
}
```

```
}
```

```
},
```

```
"offset": {
```

```
 "type": "long"
```

```
},
```

```
"source": {
```

```
 "type": "keyword",
```

```
 "ignore_above": 1024
```

```
},
```

```
"tags": {
```

```
 "type": "keyword",
```

```
 "ignore_above": 1024
```

```
},
```

```
"type": {
```

```
 "type": "keyword",
 "ignore_above": 1024
 }
}
}
}
}
}
}
```

## L1P\_Index population

The L1P\_Index is populated by the Logstash component of the ELK Stack. The following files (log-pipeline.txt, filebeat.yml) found at GitHub's interop-elk repo contains a Logstash pipeline used to populate the custom L1P\_Index elasticsearch index.

log-pipeline.txt

```
input {
 beats {
 port => "5043"
 }
}

filter {
 # if the beat is from modusbox

 mutate {
 rename => {"@timestamp" => "[beat][processing_timestamp]"}
 }

 grok {
```

```

match => { "message" =>
 "%{TIMESTAMP_ISO8601:log_timestamp} %{SPACE}%{LOGLEVEL}%{SPACE}%{SYSLOG5424PRINTASCII}%
 %{SPACE}%{PROG:log_source} %{SPACE}(.L1P_TRACE_ID=)?(%{UUID:l1p_trace_id})?(.(L1P_METRIC_TIMER:[(?
 %{JAVACLASS})][(?%{NUMBER})]|L1P_METRIC_COUNTER:[(?%{JAVACLASS})]|L1P_METRIC_GAUGE:[(?
 %{JAVACLASS})][(?%{NUMBER})]))?.*"
}

date {

 match => ["log_timestamp", "ISO8601"]

 remove_field => ["log_timestamp", "log_source"]

}

if the beat is from ripple

}

output {

 stdout { codec => rubydebug }

 if "metric" not in [tags] {

 elasticsearch{

 host => "fix_me"

 cluster => "change_me"

 protocol => "http"

 index => "l1p_index_%{+YYYY.MM.dd}"

 document_type => "l1p_log"

 }

 }

}

filebeat.yml

```

# Filebeat prospectors

---

filebeat.prospectors:

**Each - is a prospector. Most options can be set at the prospector level, so you can use different prospectors for various configurations.**

**Below are the prospector specific configurations.**

- input\_type: log

# Paths that should be crawled and fetched. Glob based paths.

paths:

```

#- /Users/honainkhan/dev/mbox/bmgf/interop-elk/filebeat/log-samples/modusbox/interop-spsp-clientproxy.log
- /home/ec2-user/elkwork/logs/interop-spsp-backend-services.log

#- /var/log/*.log

#- c:\programdata\elasticsearch\logs*

Exclude lines. A list of regular expressions to match. It drops the lines that are
matching any regular expression from the list.

#exclude_lines: ["^DBG"]

Include lines. A list of regular expressions to match. It exports the lines that are
matching any regular expression from the list.

#include_lines: ["^ERR", "^WARN"]

Exclude files. A list of regular expressions to match. Filebeat drops the files that
are matching any regular expression from the list. By default, no files are dropped.

#exclude_files: [".gz$"]

Optional additional fields. These field can be freely picked
to add additional information to the crawled log files for filtering

#fields:

level: debug

review: 1

Multiline options

Mutiline can be used for log messages spanning multiple lines. This is common
for Java Stack Traces or C-Line Continuation

The regexp Pattern that has to be matched. The example pattern matches all lines starting with [
multiline.pattern: '^[[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}'

Defines if the pattern set under pattern should be negated or not. Default is false.

multiline.negate: true

Match can be set to "after" or "before". It is used to define if lines should be append to a pattern

```

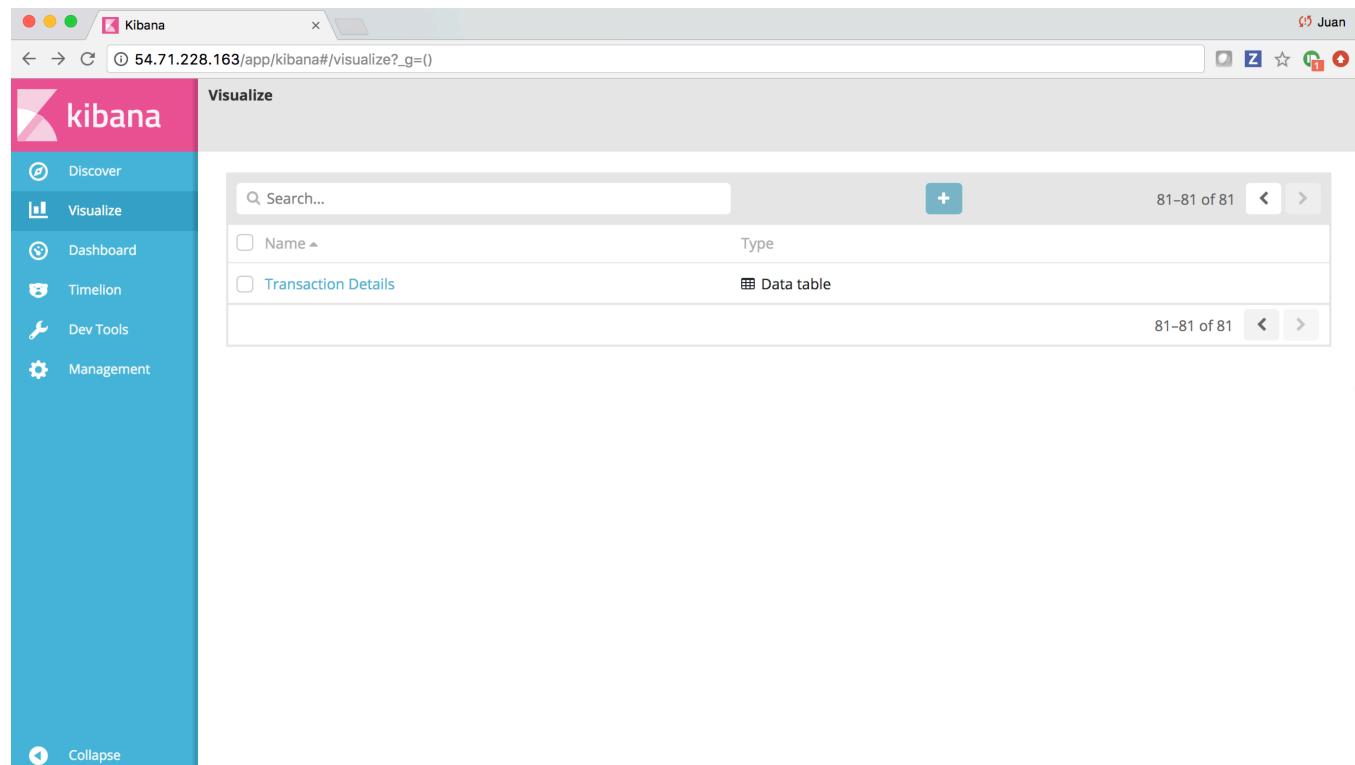
# that was (not) matched before or after or as long as a pattern is not matched based on negate.

# Note: After is the equivalent to previous and before is the equivalent to to next in Logstash

multiline.match: after

# Transaction Details Kibana Custom Visualization

In order to access the Transaction Details Kibana custom visualization navigate to Visualize menu and look for "Transaction Details".



The screenshot shows the Kibana interface with the title bar 'Kibana' and a user 'Juan'. The left sidebar has tabs for Discover, Visualize (which is selected), Dashboard, Timeline, Dev Tools, and Management. The main area is titled 'Visualize' and contains a search bar with 'Search...' placeholder text. Below the search bar are two rows of visualization cards. The first row has a card for 'Name' (Type) and a card for 'Transaction Details' (Data table). The second row is partially visible. At the bottom right of the main area, there are navigation buttons for '81-81 of 81'.

The Transaction Details Visualization shows a specific transaction identified by its L1P\_Trace\_Id across L1P components and it displays the transaction start and transaction end timestamps.

The screenshot shows the Kibana interface with the 'Visualize' tab selected. On the left, there's a sidebar with links to Discover, Visualize, Dashboard, Timeline, Dev Tools, and Management. The main area is titled 'Visualize / Transaction Details'. It displays two tables of data from the 'l1p\_index\*' index pattern.

**Table 1: de6366b6-b1d3-403b-b158-8fdbf91450df: L1P Trace Times By Service**

l1p_service_id.keyword: Descending	l1p_environment.keyword: Descending	transaction_start	transaction_end
dfsp-api	dfsp2-test	June 5th 2017, 08:51:24.522	June 5th 2017, 08:51:24.580
interop-spss-backend-services	dfsp2-test	June 5th 2017, 08:51:24.518	June 5th 2017, 08:51:24.620

**Table 2: 4e19e6a5-98cf-4c11-a1dc-d8873e8fab11: L1P Trace Times By Service**

l1p_service_id.keyword: Descending	l1p_environment.keyword: Descending	transaction_start	transaction_end
dfsp-api	dfsp2-test	June 5th 2017, 08:51:24.427	June 5th 2017, 08:51:24.504
interop-spss-backend-services	dfsp2-test	June 5th 2017, 08:51:24.422	June 5th 2017, 08:51:24.507

The Transaction Details visualization is backed by the L1P\_Index data structure.

To create the visualization navigate to Visualize menu in Kibana, select the type of visualization, select the Index (L1P\_Index), select and organize data depending on type of visualization selected and save visualization.

# Proposed PROD Architecture

Elasticsearch

Security

# Additional Considerations

Utilize Filebeat given its lightweight nature compared to Logstash. Its part of the ELK stack. Use Filebeat to ship and centralize logs. Filebeat will feed Logstash. Logstash can still be used to transform or enrich your logs and files.

Take full advantage of the Beat log shippers. Along with Filebeat, use Metricsbeat, Packetbeat and Heartbeat to monitor additional aspects of the system.

Metricbeat by default ships system metrics to elasticsearch, but there are other Metricsbeat modules that can be configured to monitor databases, http servers, queues, docker, plus any custom built modules.

Utilize a Queue in the ELK architecture before Logstash to avoid overutilization of Elasticsearch and to perform eventual Elasticsearch upgrades without losing any data during downtime.

High Availability, leverage a highly available queuing system from which Logstash servers read. Elasticsearch cluster with three master nodes.

Elasticsearch Scalability. Understand requirements and research elasticsearch accordingly.

Data Curation. Use a Curator on a cron job to delete old indices to avoid an elasticsearch crash. Also, optimize older indices to improve elasticsearch performance.

Conflict Mapping. Mapping is like a database schema in Elasticsearch. Research if this is a concern.

Security with Multi-User & Role-Based access. Understand requirements and research options.

Log Shipping. Leverage Logstash pull module to periodically go to Mule and other servers and pull data.

Log Parsing. Document grok expression used by Logstash to parse all different log types involved in L1P.

Alerting framework. Identify requirement. One can be build using cron jobs that query and generate emails based on search results.

Log archiving. Identify requirement in terms of how long to retain logs for. Also, identify storage option, e.g. S3.

ELK Monitoring. Nagios can be used to monitor the ELK stack. Nagios has some plugins to monitor Elasticseach. Also, need to monitor queue size of queuing system and health of Logstash and Kibana applications.

Logstash plugins. Beats (Filebeat), Grok, Logstash Codecs (json to plain text and vice versa), Kafka.

Keep log data protected from unauthorized access. Open Source ELK does not provide role-based access.

Maintenance requirements. Data retention policies, upgrade, etc.

Logstash and Elasticsearch should run on different machines as they both use the JVM and consume large amounts of memory. Cluster Elasticsearch, use at least 3 master nodes and at least 2 data nodes. "We recommend clustering Elasticsearch with at least three master nodes because of the common occurrence of split brain, which is essentially a dispute between two nodes regarding which one is actually the master. As a result, using three master nodes prevents split brain from happening. As far as the data nodes go, we recommend having at least two data nodes so that your data is replicated at least once. This results in a minimum of five nodes: the three master nodes can be small machines, and the two data nodes need to be scaled on solid machines with very fast storage and a large capacity for memory."

# Using Kibana

How to use Kibana, its dashboards and query language for L1P tracing and debugging purposes.

Kibana is the ELK Stack (Elastic Stack) window into the Elasticsearch data. It allows you to monitor, query, visualize and create reports on Elasticsearch data. This document is a L1P Kibana User Guide that will show how to use Kibana for the most common L1P use cases. Kibana features used at L1P projects will be described and then specific use cases will be described in detail.

## Kibana Dashboards and Visualizations

Kibana allows you to create visualizations based on the Elasticsearch data. Furthermore, Kibana allows you to create dashboards based on one or more visualizations.

Elasticsearch data is generated by the several components:

- Logstash
- Filebeat
- Metricbeat
- Heartbeat

For the data generated by the Beats family of shippers Kibana also contains out of the box Dashboards and Visualizations. These come prepackaged as part of the given Beat family shipper. After installing the given Beats family shipper its dashboards and visualizations can be imported.

## Dashboards Import

Following are the scripts to install the Beats family shipper dashboards:

- `/usr/share/filebeat/scripts/import_dashboards`
- `/usr/share/metricbeat/scripts/import_dashboards`
- `/usr/share/heartbeat/scripts/import_dashboards`

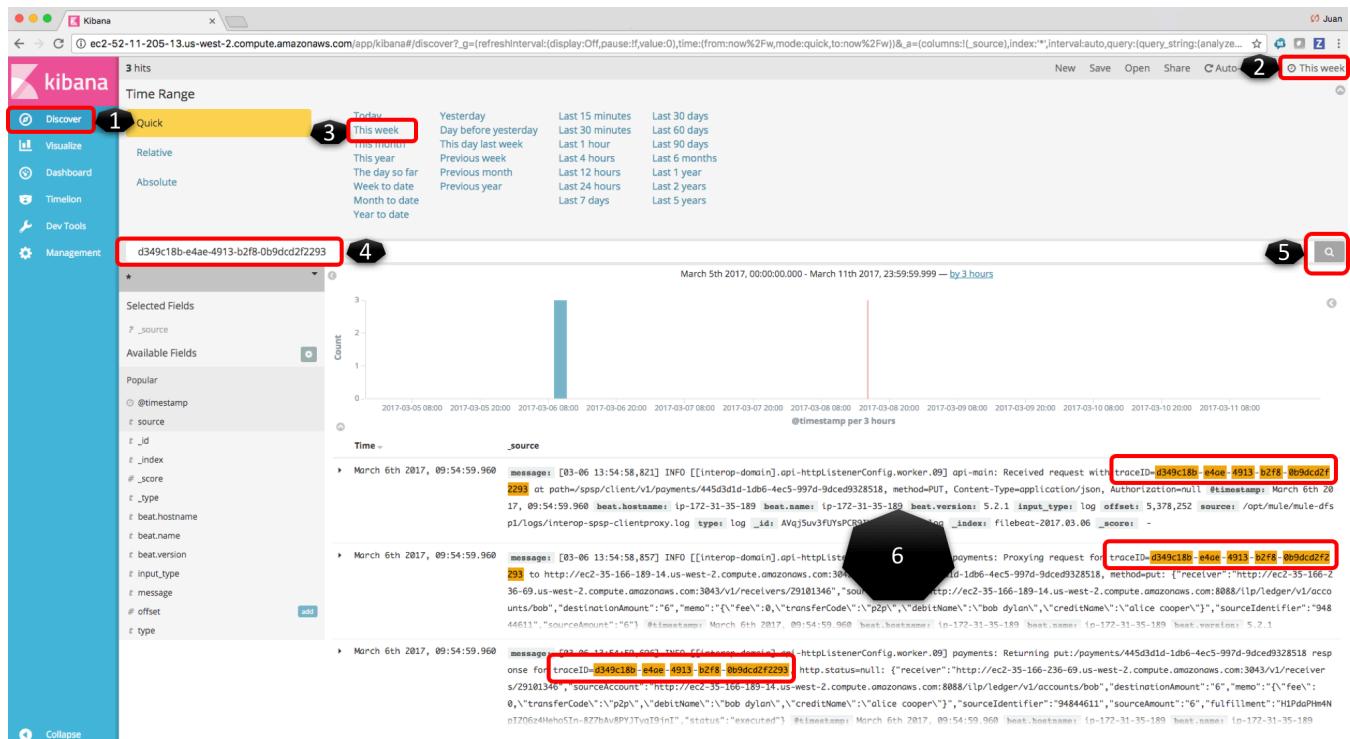
In order to access the visualization and dashboards that are imported by these scripts, go to Kibana and navigate to the Visualizations or Dashboards menu options at the left hand menu.

## Access to Kibana via NGINX on your browser

http://EC2\_INSTANCE\_URL

## L1P Kibana Use Cases

### How to monitor logs?



Follow the following steps:

1. Navigate to "Discover" menu
2. Expand Time Range, by clicking the "Time picker" icon on top right corner
3. Set Time Range, pick between Quick, Relative and Absolute modes and set time range
4. Enter your search criteria (e.g. L1p-Trace-Id)
5. Perform search
6. Review logs returned

## How to set time range?

The screenshots illustrate the process of setting a time range in Kibana:

- Top Screenshot:** Shows the main Kibana interface with a bar chart titled "May 8th 2017, 07:52:18.818 - May 8th 2017, 08:07:18.818 — by 30 seconds". The top right corner features a "Time" picker with a red box highlighting the "Last 15 minutes" button.
- Middle Screenshot:** A detailed view of the "Time Range" control panel. It includes a "Quick" section with "Today", "This week", etc.; a "Relative" section with "Last 15 minutes", "Last 30 days", etc.; and an "Absolute" section with "From: May 8th 2017, 08:03:10.771" and "To: Now".
- Bottom Screenshot:** Another view of the "Time Range" control panel, focusing on the "Absolute" section where specific dates and times are selected for the "From" and "To" fields.

By default, Kibana will show you logs for the last 15 min. To set the time range from the Discover Kibana page click on the area where with the "Time picker" icon on the top right corner. This will expand the "Time Range" control panel

down containing three different modes to set the time range; "Quick", "Relative" and "Absolute" as shown in screenshots below.

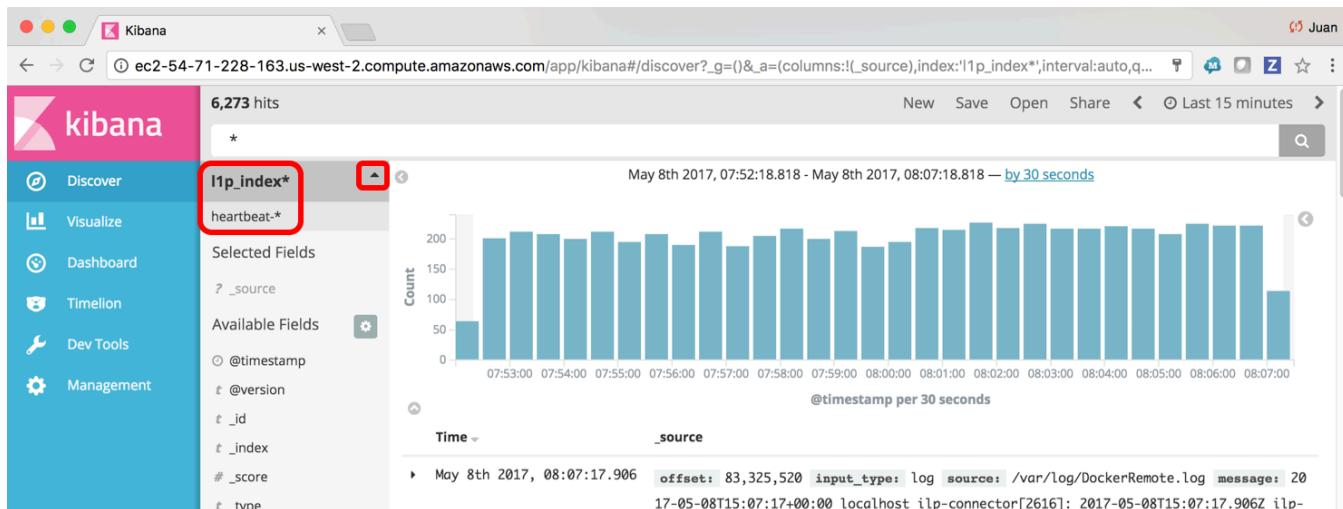
## How to enable auto refresh of search results?



The screenshot shows the Kibana Discover interface. At the top, there is a toolbar with buttons for New, Save, Open, Share, and a red box around the 'Auto-refresh' button. Below the toolbar, the search bar shows 'l1p\_index\*' and the date range 'May 2nd 2017, 00:14:48.366 - May 9th 2017, 00:14:48.366 — by 3 hours'. On the left, there is a sidebar with navigation links: Discover (selected), Visualize, Dashboard, Timelion, and Dev Tools. The main content area displays a table of search results with 1,786,700 hits. A dropdown menu titled 'Refresh Interval' is open, showing options from 'Off' to '1 day'. The 'Off' option is highlighted with a red box.

Search results can be set to auto refresh, so your search results and visualizations do not contain stale data. Optionally, you can manually refresh results by clicking "Refresh". The Auto-refresh can be enabled by clicking the "Time picker" icon, clicking the "Auto-refresh" link and then set it to on and specify the refresh rate.

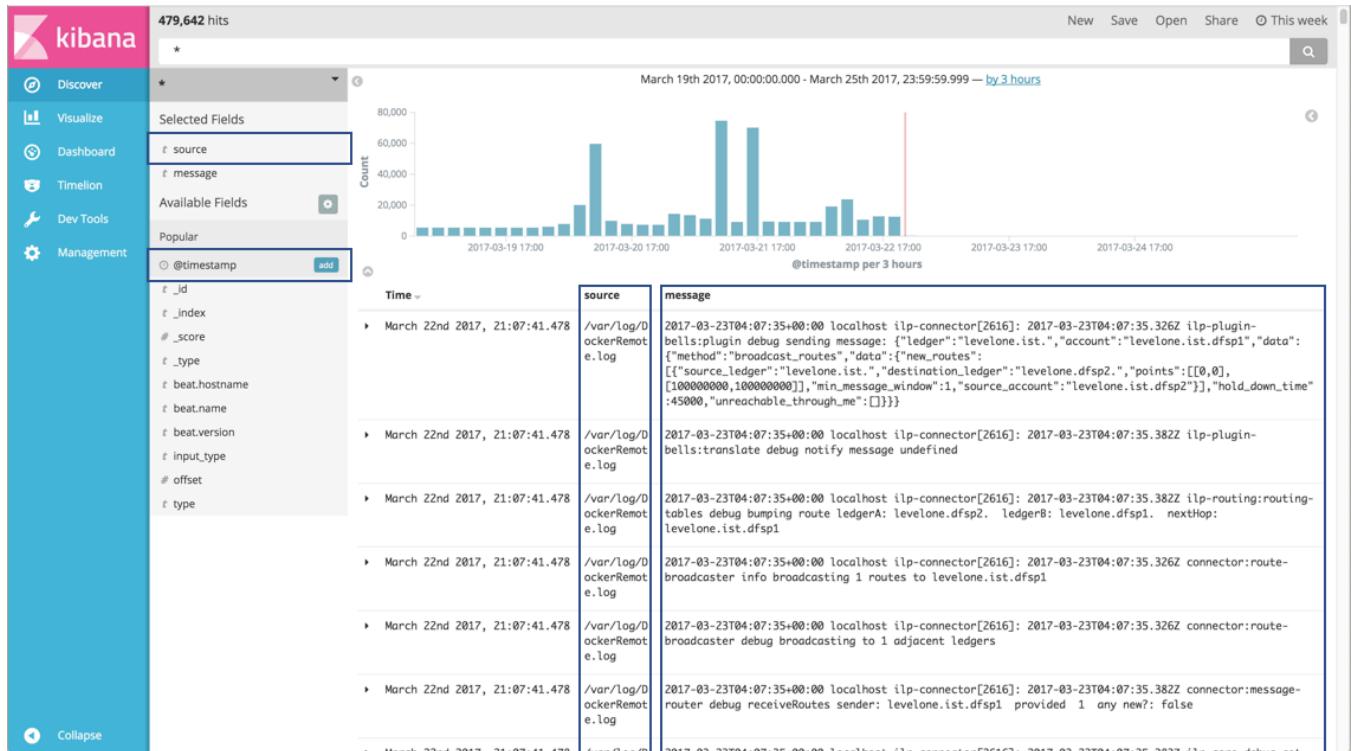
## How change to which indices you are searching?



The screenshot shows the Kibana Discover interface. The sidebar on the left has 'Discover' selected. In the main content area, the search bar shows 'l1p\_index\*' and the date range 'May 8th 2017, 07:52:18.818 - May 8th 2017, 08:07:18.818 — by 30 seconds'. Below the search bar, the index pattern 'l1p\_index\*' is highlighted with a red box. To its right, there is a dropdown arrow icon. The main content area displays a bar chart showing the count of events over time. The x-axis is labeled '@timestamp per 30 seconds' and the y-axis is labeled 'Count'. At the bottom, there is a detailed log entry: 'May 8th 2017, 08:07:17.906 offset: 83,325,520 input\_type: log source: /var/log/DockerRemote.log message: 2017-05-08T15:07:17+00:00 localhost ilp-connector[2616]: 2017-05-08T15:07:17.906Z ilp-

When you submit a search request, the indices that match the currently-selected index pattern are searched. The current index pattern is shown below the toolbar. To change which indices you are searching, click the index pattern and select a different index pattern. NOTE: By default, only 1 index pattern is shown, you must click the arrow to expand the index section to show you the different indexes available.

## How to add/remove fields from Kibana's Discover window to monitor logs?



Navigate to the Kibana Discover page and hover over the field you would like to add/remove from the search results table and click the add/remove button.

## How to trace a L1P transaction based on its L1p-Trace-Id?

Navigate to the Kibana Discover page and enter a search criterial like:

L1p-Trace-Id=d349c18b-e4ea-4913-b2f8-0b9dcd2f2293

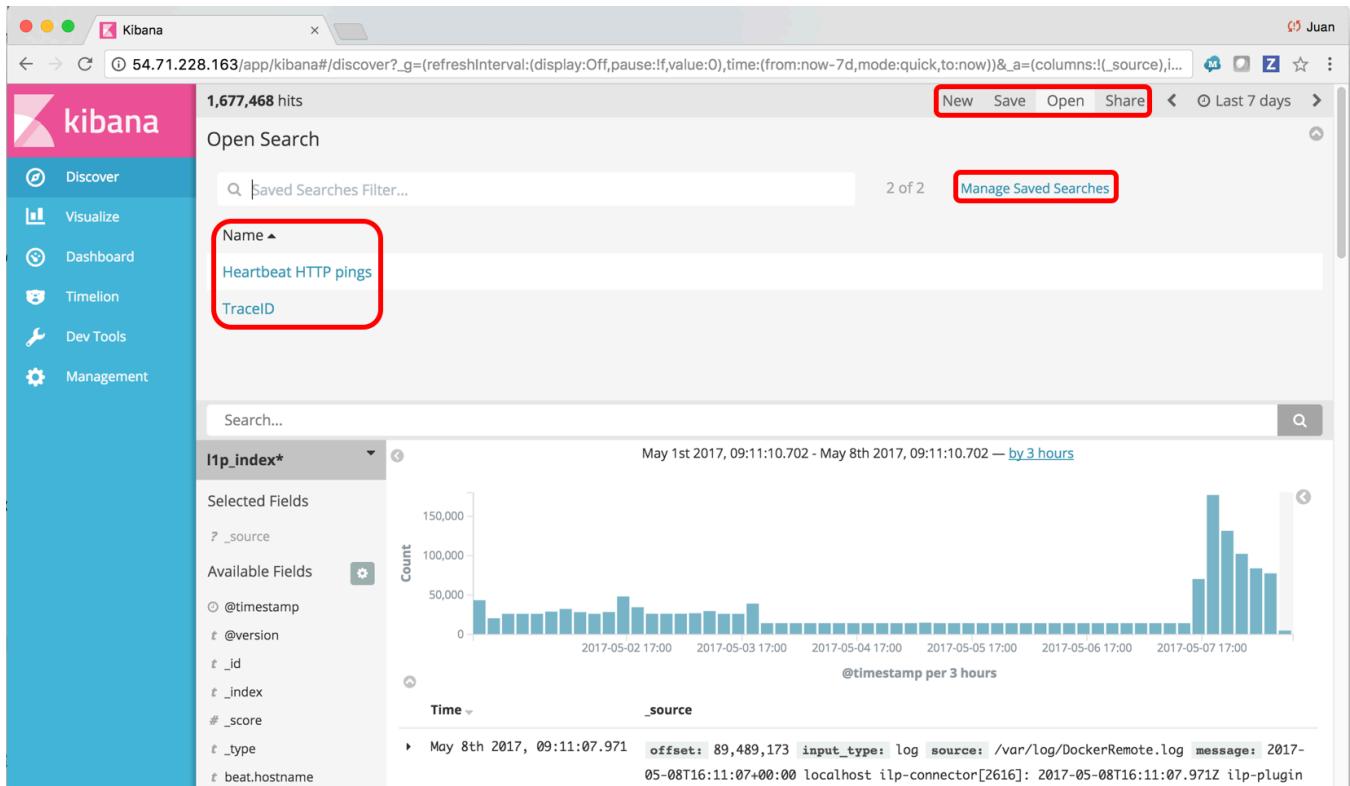
## Examples of common L1P Kibana Queries

Sample Query	Description
<code>l1p_trace_id:"47eb4790-0ccd-487a-9f05-f239ecf5d8a2"</code>	Search for a specific L1P-Trace-Id
<code>"89969d15-582a-4fee-9d2a-0dc732acc130"</code>	Search for a specific L1P-Trace-Id
<code>_exists_:l1p_trace_id</code>	Search for log entries that contain a l1p_trace_id value
<code>_index:"l1p_index_2017.04.18"</code>	Search for a specific Index
<code>l1p_trace_id:"1614cfa4-e792-4b01-9537-2f3eb8001b5e" AND _index:"l1p_index_2017.06.28"</code>	Search for a specific L1P-Trace-Id <b>AND</b> inside a specific index

## Quering behaviour and rules in Kiabana:

- query behaves as unstructured text search, with some special commands, and if you get the command syntax wrong it just does an unstructured text search
- by default it searches for entries containing any or your search terms
- hyphen is consider as a delimiter
- in order to search for a string literal use double quotes NOT single quotes (single quotes are ignored)
- to search for a single filed enter field name, then a colon and then the value within double quotes
- exists and missing are examples of commands that can be used, e.g exists:exception
- AND and OR are case-sendiftive, must use upper case
- can use parenthesis when searching for several things, e.g. exists:exception AND ( `l1p_trace_id:"1614cfa4-e792-4b01-9537-2f3eb8001b5e"` OR `_payment_id:"1614cfa4-e792-4b01-9537-2f3eb8001b5e"` )
- field names are also case-sensitive, examples of fields are `l1p_trace_id` and `index`

## How to save a search, how to open a saved search and how to manage saved searches?



The screenshot shows the Kibana Discover interface. On the left, there's a sidebar with icons for Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The main area has a pink header bar with the Kibana logo and a search bar labeled "Saved Searches Filter...". Below the header, it says "1,677,468 hits" and "Open Search". At the top right, there are buttons for "New", "Save", "Open", "Share", and a time range selector "Last 7 days". A red box highlights the "Save" button. Below this, there's a table with a single row: "Name ▲ Heartbeat HTTP pings" and "TraceID". Another red box highlights the "Heartbeat HTTP pings" entry. To the right of the table is a histogram titled "May 1st 2017, 09:11:10.702 - May 8th 2017, 09:11:10.702 — by 3 hours". The x-axis is "timestamp per 3 hours" and the y-axis is "Count". The histogram shows a significant peak around May 7th. Below the histogram, there's a table with columns "Time" and "source". A specific entry is expanded: "May 8th 2017, 09:11:07.971 offset: 89,489,173 input\_type: log source: /var/log/DockerRemote.log message: 2017-05-08T16:11:07+00:00 localhost ip-connector[2616]: 2017-05-08T16:11:07.971Z ilp-plugin". A red box highlights this expanded entry.

Kibana allows you to save a search criteria. From the Kibana Discover page just hit the "Save" link on the top right hand corner just before the "Time Picker" icon to save your search. To access your saved search, hit the "Open" link. To delete or edit your save search hit the "Open" link and then the "Manage Saved Searches" link.

## How to monitor transaction duration between l1p\_components?

The screenshot shows the Kibana Visualize interface with a sidebar on the left containing links for Discover, Visualize, Dashboard, Timelion, Dev Tools, and Management. The main area is titled "Visualize / Transaction Details".

**l1p\_index\*** (Selected)

**metrics**

- Metric: Min @timestamp
- Metric: Max @timestamp

**buckets**

- Split Tab...: l1p\_trace\_id.keyword: Descending
- Split Ro...: l1p\_service\_id.keyword: Descending
- Split Ro...: l1p\_environment.keyword: Descending

**75000825-7b1e-4428-8df9-7fa7b3e04032: L1P Trace Times By Service**

l1p_service_id.keyword:	l1p_environment.keyword:	transaction_start	transaction_end
interop-dfsp-directory	dfsp1-test	May 3rd 2017, 20:21:03.588	May 3rd 2017, 20:21:03.588

**db02dd89-8fc9-458c-b9a5-53f69d80f31e: L1P Trace Times By Service**

l1p_service_id.keyword:	l1p_environment.keyword:	transaction_start	transaction_end
interop-spsp-backend-services	dfsp1-test	May 3rd 2017, 20:21:03.588	May 3rd 2017, 20:21:03.588

A Kibana Visualization on top of the custom l1p\_index was created for this purpose and can be accessed by navigating to Kibana Visualize page and selecting the "Transaction Details" link. NOTE: Once the trace id is contained at all component and services logs this table will correctly display the durations.

## Reference to Kibana's Official Documentation

<https://www.elastic.co/guide/en/kibana/current/introduction.html>

# Using Logging

## Introduction

This document provides Mojaloop services logging guidelines in order to provide end-end traceability of interactions, aid in troubleshooting and publish metrics to the backend

### Desired Goals

- End-to-end Traceability of a particular transaction
- Understand service behavior
- Debugging
- Metrics for a particular transaction

### General:

All logs statement must begin with **ISO8601 compliant timestamp**. The timestamp must have millisecond resolution. It should be followed by a log level. Available logs levels are **ERROR, WARN, INFO, DEBUG** For example

```
2017-04-28T17:16:20.561Z INFO ilp-routing:routing-tables debug
bumping route ledgerA: mojaloop.dfsp1. ledgerB: nextHop:
mojaloop.ist.dfsp2
```

### 1. End-to-end Traceability:

To provide end-to-end traceability of Level One payment related interactions, L1P components shall include **L1p-Trace-Id** in all log statements where available. The following snippet must be included in all of the log lines: **L1p-Trace-Id=<current\_trace\_id>** For a given unit of work, related interactions between services, the L1p-Trace-Id is required to be unique. It is recommended that UUID be used for uniqueness requirement. This would allow to quickly retrieve all logging for a given L1p-Trace-Id.

## 2. Rest Service Calls:

All Rest Service calls must include **L1p-Trace-Id** as a header HTTP Header. The value of this header must be set to *Payment ID* for all payment interactions. For all non-payment interaction the originating DFSP must generate and use an UUID for the value of the header. In the case where the **L1p-Trace-Id** is not present as a header, an error needs to be logged with context about the call with the missing header. The service must set the **L1p-Trace-Id** header with appropriate value whether the current interaction is during the course of processing a payment or not.

## 3. Web Socket Notifications:

[To Be Filled in]

## 4. Additional Context in Logs

It is recommended to log other identifiers that can help retrieve logs statement across multiple layers in the Mojaloop stack. Some examples are Transfer Id, User ID (USSD id, email, login name), AppName, and AccountId etc.

Relevant-Id=<id\_value>

## 5. Metrics Logging

Logs can be used to publish metrics to metrics service. There are 2 types of metrics that are supported. The details about supported metrics are available [here](#). The snippet below shows the syntax for publishing metrics:

1. Counter . . . L1P\_METRIC\_COUNTER: [counter-namespace.name] . . . where L1P\_METRIC\_COUNTER is a keyword followed by a colon and the desired metric name\* is within [ ]. This would increment the counter identified by metric name by 1.
2. Timer . . . L1P\_METRIC\_TIMER: [timer-namespace.name] [50] . . . where L1P\_METRIC\_TIMER is a keyword followed by a colon, the desired metric name is within [ ] and timed value in millisconds with [ ].  
*This would add the time value in milliseconds to timer identified by metric name.*

\* metric name is composed of 3 elements that separated by a period.

1. environment which captures where the application is running e.g. *dfsp1-test*, *dfsp2-qa* etc. 1. application instance id which should identify the process 1. metric name which could contain alphanumeric characters and could have java package name style prefix

# Forensic Logging Sidecar

The sidecar is a service that acts as an intermediary to facilitate messaging between the services and central kms, including the following functions:

- Relaying a batch of events/messages from the connected services to the central kms
- Performing a health check on the sidecar

The following documentation represents the services and endpoints responsible for various sidecar functions.

Contents:

- [Deployment](#)
- [Configuration](#)
- [Connect And Write](#)
- [Logging](#)
- [Tests](#)

## Deployment

### Deploy to Docker

To deploy the sidecar with a service in a Docker container, create a Docker compose file at the root level of the service and add the following as its content:

```
version: '2'
services:
 central-ledger:
 build:
 context: ./
 dockerfile: api.Dockerfile
 environment:
```

```

 CLEDG_DATABASE_URI:
"postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@postgres:5432/
central_ledger"
 CLEDG_SIDECAR__HOST: "forensic-logging-sidecar"
networks:
- sidecar
depends_on:
- postgres
- forensic-logging-sidecar
central-ledger-admin:
build:
context: ./
dockerfile: admin.Dockerfile
environment:
 CLEDG_DATABASE_URI:
"postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@postgres:5432/
central_ledger"
 CLEDG_SIDECAR__HOST: "forensic-logging-sidecar"
networks:
- sidecar
depends_on:
- postgres
- forensic-logging-sidecar
forensic-logging-sidecar:
env_file: .env
environment:
 SIDE_DATABASE_URI:
"postgres://${SIDE_POSTGRES_USER}:${SIDE_POSTGRES_PASSWORD}@postgres:
5432/sidecar"
 SIDE_KMS_URL: "ws://kms:8080/sidecar"
image: modusbox-level1-docker-release.jfrog.io/@mojaloop/
forensic-logging-sidecar:latest
depends_on:
- postgres
- kms
networks:
- back
- sidecar
kms:
env_file: .env
image: modusbox-level1-docker-release.jfrog.io/@mojaloop/
central-kms:latest
depends_on:
- postgres
networks:
- back
networks:

```

```
sidecar: {}
```

## Deploy to ECS

To deploy the sidecar with a service to ECS, make sure you've created an account on AWS and use this as an example for deploying the service and sidecar to ECS:

```
'use strict'

const AWS = require('aws-sdk')
const ecs = new AWS.ECS()
const Variables = require('./variables')

const registerTaskDefinition = (name, service, image, port,
environment = []) => {
 const params = {
 containerDefinitions: [
 {
 name,
 image,
 essential: true,
 memoryReservation: 200,
 portMappings: [
 {
 containerPort: port
 }
],
 links: [
 `${Variables.SIDECAR.NAME}:${Variables.SIDECAR.NAME}`
],
 environment,
 logConfiguration: {
 logDriver: 'syslog',
 options: {
 'syslog-address': 'tcp://127.0.0.1:514',
 'syslog-facility': 'daemon',
 'tag': 'central-ledger'
 }
 }
 },
 {
 name: Variables.SIDECAR.NAME,
 image:
`${Variables.AWS_ACCOUNT_ID}.dkr.ecr.${Variables.AWS_REGION}.amazonaws.com/${Variables.SIDECAR.IMAGE}:latest`,
```

```

 essential: true,
 memoryReservation: 200,
 portMappings: [
 {
 containerPort: Variables.SIDECAR.PORT
 }
],
 environment: [
 {
 name: 'SIDE_SERVICE',
 value: service
 },
 {
 name: 'SIDE_DATABASE_URI',
 value:
`postgres://${Variables.SIDECAR.POSTGRES_USER}:${Variables.SIDECAR.PO
STGRES_PASSWORD}@${Variables.SIDECAR.POSTGRES_HOST}:5432/sidecar`
 },
 {
 name: 'SIDE_KMS_URL',
 value: Variables.SIDECAR.KMS_URL
 }
]
],
 family: name
}

return ecs.registerTaskDefinition(params).promise()
.then(result => {
 const revision = result.taskDefinition.taskDefinitionArn
 console.log(`Registered task definition: ${revision}`)
 return revision
})
}

const deployService = (clusterName, service, taskDefinition) => {
 const params = {
 service,
 taskDefinition,
 cluster: clusterName,
 desiredCount: 1
 }

 console.log(`Deploying ${taskDefinition} to ${service} on cluster
${clusterName}`)
}

```

```

 return ecs.updateService(params).promise()
 .then(result => result.service.taskDefinition)
 }

module.exports = {
 registerTaskDefinition,
 deployService
}

```

*For more guidance on deployment to ECR, look in the deploy folder at the root of the project.*

# Configuration

## Environment variables

The Sidecar has a handful of options that can be configured through environment variables.

Environment variable	Description	Example values
SIDE_DATABASE_URI	The connection string for the database the sidecar will use. Postgres is currently the only supported database.	postgres://:@localhost:5678/forensic_logging_sidecar
SIDE_SERVICE	The name for the sidecar service.	TestService
SIDE_BATCH_SIZE	The message batch size.	64
SIDE_PORT	The port the sidecar server will run on.	5678
SIDE_KMS_URL	The url for the KMS	ws://localhost:8080/sidecar
SIDE_KMS_PING_INTERVAL	The time, in milliseconds, between pings to the KMS	30000
SIDE_KMS_REQUEST_TIMEOUT	The time, in milliseconds, to timeout a request to the KMS	30000
SIDE_KMS_CONNECT_TIMEOUT	The time, in milliseconds, to timeout a connection attempt to the KMS	60000
SIDE_KMS_RECONNECT_INTERVAL	The time, in milliseconds, between connection attempts to the KMS	5000

# Connect And Write

For our examples we've written client.js which extends EventEmitter to connect and write to the sidecar.

Once the sidecar has launched, your client should connect to it using the exposed port.

```
connect (port) {
 this._client = Net.createConnection({ port }, () => {
 this.emit('open')
 })

 this._client.on('data', data => {
 this.emit('data', data)
 })

 this._client.on('end', () => this.emit('end'))
}
```

After connecting to the sidecar, the client can write to it.

*The message should be prefixed with the length of the message written as a big endian. The length should be a valid unsigned 32-bit integer*

```
write (msg) {
 let message = Buffer.isBuffer(msg) ? msg : Buffer.from(msg)
 let length = message.length

 let buffer = Buffer.alloc(this._prefixSize + length)
 buffer.writeUInt32BE(length, 0)
 message.copy(buffer, this._prefixSize)
 this._client.write(buffer)
}
```

*For a more complete example, look in the examples folder at the root of the project.*

# Logging

Logs are sent to standard output by default.

# Tests

Tests include unit and functional.

Running the tests:

```
npm run test:all
```

Tests include code coverage via istanbul. See the test/ folder for testing scripts.

# Mule's Docker Image

This docker image is based on Java's official docker image.

Applications are incorporated into the image, which is why we need to change the dockerfile in order to do a COPY command to copy the zip file into mule's applications folder: "/opt/mule/apps"

Mule folders for domains, configs and logs are mounted volumes so those can be mapped to host directories.

## Building it

### Parameters

There are two different parameters to build, one is the version and the other is the port that is exposed. In case a parameter is missing then the defaults are 3.8.0 for version and 8081 for ports. The versions that are supported are: 3.8.0, 3.7.0, 3.6.1, 3.6.0 and 3.5.0

### Command

In order to build, the following command is used replacing the <> with the actual values:

```
docker build -t <>/<>:<> -f <>
<>
```

username is only required if the image needs to be pushed to docker hub. If dockerfile is named "Dockerfile" then there is no need to specify the -f <> argument. Mule's application zip should be saved into dockerBuildPath.

### Final output

After this command is run, a mule image that is ready to be instantiated into a container is created.

## Running a container from the image created

The following command starts and runs a container:

```
docker run -d -p 8081:8081 -v <>hostFolder>>:<>containerVolumeFolder>> --name <>containerName>> <>imageUserName>>/<>imageName>>:<>imageTag>>
```

-d is indicating that it will run as a daemon. -v is used to map a hostFolder to a container's volume.

"imageUserName:""imageName""imageTag" is mandatory and specifies which image to use for the new container.

### Example

```
docker run -d -p 8081:8081 -v ~/hostMuleLogs:/opt/mule/logs --name myMuleContainer modusbox/mule:latest
```

The following command can be used to log into a container:

```
docker exec -ti <>containerName>> bash
```

# Configuring High Availability Proxy

## Introduction

Load balancing across multiple server instances is one of the amazing techniques and ways for optimizing resource utilization, maximizing throughput, and reducing latency to ensure high availability of servers in an environment where some concurrent requests are in millions from users or clients and appear in a very fast and reliable manner.

This Document explains about Installing and configuring HAProxy on AWS EC2 Instance

HAProxy is free, open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers. It is written in C and has a reputation for being fast and efficient (in terms of processor and memory usage).

## Install HAProxy

Install a HA Proxy in AWS EC2 Instance by Issuing the command 'sudo yum install --enablerepo=epel haproxy'.

After the Successful Installation check the version Installed by Issuing the command 'haproxy -version'.

```
Senthilkumars-MacBook-Pro:HaProxy-Mule senthil$ haproxy
-version
HA-Proxy version 1.7.4 2017/03/27
```

Create a Config file for Mule Instances

## Sample Config file

```
listen stats
bind *:9002
mode http
log global
maxconn 10
clitimeout 100s
srvtimeout 100s
contimeout 100s
timeout queue 100s
stats enable
stats hide-version
```

```
stats refresh 30s
stats show-node
stats auth admin:mypassword
stats uri /haproxy?stats
stats admin if TRUE
frontend localnodes
 bind *:9001
 mode http
 default_backend nodes
backend nodes
 mode http
 balance roundrobin
 server DFSP1-Test ec2-52-32-100-1.us-
west-2.compute.amazonaws.com:8088 maxconn 100 check
 server DFSP2-Test ec2-52-32-200-2.us-
west-2.compute.amazonaws.com:8088 maxconn 100 check
```

After creating the configuration file for the Mule Instance place it under any directory (Eg: /home/ec2-user/scripts/modusbox/haproxy),

Execute the following command from the directory to run the load balancer configuration

```
haproxy -f haproxy_mule.cfg
```

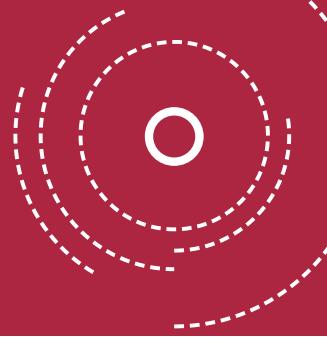
After the successful execution, Now you will be able access the Mule APIs with the URL

```
http://<AWS ec2 Instance>:9001/
```

Status of the load balancer can viewed thru the browser under the path

```
http://<AWS ec2 Instance>:9002/
```

# Testing



These tests describe the expected behavior of the services separate from the interfaces. Should the interfaces change, these behaviors would be expected to continue to work. Each test is described as a simple bulleted sentence that states the test conditions and expected behavior. This has the same information you might expect in a more formal format like Gherkin, but is easier to read and review.

# Account Management Tests

Account management is internal to the DFSP service. It is tested and verified via the USSD interface.

## Use Cases

### Add account

A user can have zero or more accounts. The first account is made when the user signs up with the DFSP (see user management tests). Adding an account will add additional accounts for that user in the same DFSP.

The Account Name and Is Primary (Y/N) are required inputs. When a new account is created it shows up in the switch accounts list.

Currently, the account name is unique within DFSP. You can't create an identical account name for two different users. This is a convenience for the current implementation, and not tested. This restriction can be removed without loss of functionality.

### Close account

- Secondary accounts can be removed.
- A non-signatory doesn't have the option to close an account. This is manually verified.
- The primary account can't be closed (gives an error message when you try).
- An account can't be closed if there is money in it. There's an error message when it's attempted.
- The account signatory can close an empty account.

### Switch account

- USSD shows the message: "You don't have any other account to switch to" when there is only

one account.

- A user can switch between available accounts when there is more than one.

## Primary (default) Account

- When there is more than one account, the primary account is where money will be sent.
- A new account can be made primary when it is created. The previous primary account will become a secondary account.
- A secondary account can be made primary. The previous primary account will become a secondary account.

## Additional holders (users)

The original holder for the account is considered the owner. For now, there is no option to change owners. Additional holders can be signatory or non-signatory.

### Add additional holder to account

- Non-signatory can't add a user. There's no option for this and it is manually verified.
- A signatory for the account can add other users to an account as a holder of that account. If a user is added to someone else's account, that user can see the new account in their list of accounts to switch to. They can switch to the account and do things like see the balance.

### Remove additional holder from account

- The original owner (first user) of the account can't be removed. (requires remove user from DFSP). This gives an error when attempted.

- A non-signatory can't remove anyone. There is no option to do this and it is manually verified.
- A signatory can remove another user from an account who is not the owner.

## Account Permissions

- The owner is a signatory and they have same menus (manually verified). An owner can't be made non-signatory (error).
- A signatory can change another holder, who is not an owner, from signatory to non-signatory and vice-versa.
- A non-signatory holder can't send money or manage accounts. They can send invoices, look at the mini-statement, and account balance (manually verified).

# Customer Management Tests

## Association of a phone to a customer

A phone has an identifier that the DFSP uses to associate the phone with a customer.

- If a customer connects using a phone with an unassociated identifier then the customer is asked to create an account.
- If the customer connects with a phone that has been associated with a customer, the customer receives a menu of account actions.

There is currently no way to associate another phone with the same customer.

## Add customer

A customer is identified by at least their name, birthdate, and an ID such as their national ID number. Adding a customer requires this data and returns the user number from the central directory service. Adding a customer creates at least one account for that customer in the DFSP.

- If customer already registered at that DFSP, then attempting to add the customer again from another phone (same name, birthdate, and ID) returns an error.

- If the fraud service returns 100, customer isn't added (error).
- A new customer can add themselves to the DFSP. The customer is registered in central directory associated to that DFSP.
- Different customers can be registered with same name and/or birthdate.
- The same customer can be registered with multiple DFSPs, though they will have different customer numbers.

## Remove customer

- If the account owner closes the last account (see account management), the customer account closed at DFSP, and no customers can connect to that account. The customer is no longer associated with the DFSP in central directory and the phone will again ask for an account to be created.

## Change password

Changing password functionality is not currently implemented [#420]. - Simple passwords are not allowed (all one number, straight runs, too short) [Not implemented: #331] - A customer has a single password for all accounts in a DFSP. This is a convenience for our implementation, and not tested.

## Account Types

- There are several types of accounts that may be created: customer, agent, and merchant. A

customer has only one account type and it is set when the customer is added. Currently there is no option to change account types or have different account types for a single customer.

## Merchant accounts

- Merchant can send pending transactions, others can't send pending transactions but can approve them (manual verified).

## Agent accounts

- Agents start with two accounts: the main one and the commission account.
- Commission account can't be closed (no option exists, manually verified)
- The commission account can't be made the primary (error)
- Agents have the option to do Cash In and Cash Out transfers, others don't. The commission account can't send money, cash-in, cash-out, or pending transactions (manually verified).

# DFSP Management

DFSP's can be registered and put on hold in the central directory. Doing so enables and disables the DFPS from conducting transfers with the central ledger.

These operations are done through (restful API calls to the central directory)[[https://github.com/mojaloop/Docs/blob/master/CentralDirectory/central\\_directory\\_endpoints.md](https://github.com/mojaloop/Docs/blob/master/CentralDirectory/central_directory_endpoints.md)]

## Add DFSP

Registers the DFSP with the central directory.

## Pause DFSP

Not yet implemented. This should cause all calls for that DFSPs users to return "unknown" and all pending transfers for that DFSP to be cancelled at the center. It could be called by a DFSP for itself or a regulator at the center.

## Unpause the DFSP

Not yet implemented. Would renew normal operations for the DFSP.

# Fee tests

Below we list the equivalence classes that make of the test combinations in the test matrix.

## Variations

### Fee Source

- Sender fee
- Receiver fee
- Agent cash-out fee
- Agent cash-in fee
- Central fee

### Path for transfer

- Cross-DFSP
- Same DFSP (should not apply center fee)

### Configure Amount

- Stair-step: flat fee plus percent for range
- Zero

# Test Matrix

Using pair combinations of the variations we get a matrix like this:

Path	Source	Receiver	Center	Agent Cash In	Agent Cash Out
Cross-DFSP	0	Stair-step	0	Stair-step	0
Cross-DFSP	Stair-step	0	Stair-step	0	Stair-step
Same-DFSP	Stair-step	Stair-step	0	0	Stair-step
Same-DFSP	0	0	Stair-step	Stair-step	0
*	0	*	Stair-step	0	Stair-step
*	Stair-step	*	0	Stair-step	0
Cross-DFSP	*	*	*	Stair-step	Stair-step
*	*	*	*	0	0

\* value doesn't matter

# Validations

For each variation verify that the fees can be:

- Configured - Shown to the sender (it's enough to check the quote return from the scheme adapter)
- Deducted from the transfer - Itemized for settlement (for these last two it's enough to check central ledger)

# Pending transaction tests

Merchants are able to send a pending transaction. These show in their pending transaction list till resolved. Anyone can approve or reject a pending transaction sent to them.

## Send pending transaction

Assumes the user is a merchant - (x) Send invoice for 0 - (x) Send for valid amount to non-existent customer, get error - Send pending transaction for a valid amount to valid customer

## Approve pending transaction

- Approve a transfer, the money and fees are transferred from approver's account. The principle goes to the pending transfer sender.
- (x) Approve a transfer when the amount exceeds the user's balance, get error and transaction is not sent or rejected.

## Reject pending transaction

- Reject the proposed transfer. Notification goes back to sender.

# Send Money Tests

As part of the [Level One principles](#), the customer must be able to see at least the name of the person or business they are sending their money to and the full cost of the transfer, broken out by principle and total fees, before they approve sending money. Money can only be sent (pushed) not debited (pulled).

## Variations

Instead of listing every case, we list the equivalence classes for variations that can be done when sending money. These include positive cases, positive and negative boundary cases, and invalid cases. In all positive cases, the fulfillment should be recorded in the ledgers of both the payee and payer DFSPs and the central ledger. Under no cases should the payment not be represented correctly in all three ledgers, though for some negative cases, the matching will require services to be restarted or connections to be reestablished.

Combinations of some equivalence classes should be tried. In general, negative cases, marked (x), are not combined with other variations unless mentioned and should have an error message.

### Destinations

- Payer and Payee are on the same DFSP
- Payer and Payee are on separate DFSPs
- (x) Invalid Destination customer

### Customers

Combine with destinations - Same customer - Different customer - Same customer, different account but same DFSP

### Test Matrix

This test matrix condenses the positive cases above into two simple tests. Other variations are covered below and in other tests.

Destination	Customer
Same DFSP	Same customer different account
Different DFSP	Same customer but different ID

## Amount to send

Amounts don't need to be combined with other variations. - 1 - Some - Exact account balance - (x) More than account balance due to fees - (x) More than account balance

## DFSP Limits

Limits to the number of transfers in a day, the maximum account size, or the maximum transfer amount are configuration limits implemented at the DFSP and don't need to be combined with tests of other services.

If there is a limit on the number of transfers, then a cancelled or rejected transfer still counts against a customer limit. Refunds are separate transfers initiated by the DFSP that do not count against a customer limit. Sending to yourself on the same DFSP shouldn't be counted toward any limits.

Configuration: Verify limits can be set to both none and some amount. Changing the limits should be logged to the forensic log.

Set the maximum number of transfers and transaction size low (2) and try: - The maximum number of transfers in a day - (x) One more than the maximum number of transfers per day - Send the maximum transaction size - (x) Send more than the maximum transaction size - exceeding limits to yourself (should work)

## States

A transfer can be one of several states. Some of the states have multiple ways they can occur. Each of these variations needs to be tested, but don't need to be combined with other variations.

To be able to test the cancel states we need to be able to hold the sending of a the payment messages in each service till after the timeout. Likewise, to test the rejection states we need to be able to force a rejection from the center or the DFSPs.

Here are the list of possible states:: - Unknown - Preparing and within timeout. This is part of the normal flow, it tested in regular end to end tests. - After timeout, but not notified. This happens when a service is down or a message is dropped and is tested in the resilience tests below. - Cancelled (timeout) - Payer DFSP timeout. After the quote the payer DFSP doesn't send the prepare till it has already timed out. If it attempts to send it anyway, the center should reject the prepare and the sender should show a cancellation to the customer. - Center timeout during prepare. The center sits on the payment till it times out. - Payee DFSP timeout. The receiver sends a cancel notification. - Center

timeout during fulfill. The center acknowledges the fulfill message, but sends a cancellation for the notification to both sender and receiver. - Final Payee timeout. In this state the transfer is already fulfilled. Even if the timeout occurs after receipt by the sender but before the sender ledger handles it, the sender DFSP should process the transfer. - Rejected - Payer DFSP rejects (ex: fraud or insufficient account funds) - Center rejects (ex: insufficient settlement funds) - Payee rejects (ex: fraud) - Fulfilled

See settlement tests below for additional validations.

## Thread contention/Sequence errors

- Remove a destination user after the quote but before the transfer is received by the destination DFSP. This should result in a rejection of the transfer by the Payee DFSP.

## Time skew

- Verify time skew is not relevant by setting each service on different dates and sending money. It is expected that the cross-service logs will show odd times.

## Resilience

Despite the failures listed below, no ledger should lose money and the transfer should eventually succeed when the failure is resolved. These are negative tests and not combined with other variations.

- Message failures. The failures occur when messages are dropped or not sent
  - Halt fulfillment notification messages for center
  - Halt fulfillment notification messages for payee DFSP
  - Halt fulfillment notification messages for payer DFSP. Transfer should go through due to retries when the connection is re-established.
- Service failures. In each case the payment should complete after the service is restarted.

- Take down payee DFSP after quote
- Take down center DFSP before and after prepare
- Take down payer ledger adapter after prepare
- Take down client when a transfer is unknown (is retry initiated by the DFSP when the client is restarted?)
- Verify Idempotence - cause retries and verify only 1 transaction on ledgers for both DFSPs and the center.

## Settlement

To support deferred net settlement, the central ledger can easily list:

- Fulfilled transfers
  - with fees broken out for separate accounts.
- Balances by DFSP
- Cancelled and rejected transfers
- Unknown expired transfers

The first two are tested as part of fees testing. The latter two should be tested during state testing.

## Refunds

Refunds are not currently implemented.

In this system all transfers are final, so a refund is a second transfer in the opposite direction for the original amount including both principle and fees. It contains data to link it to the original transfer it is negating for auditing purposes.

DFSPs typically do not charge fees for the refund.

The refund is marked as such so that the central ledger can report on it appropriately.

Refunds may be charged a central fee if that is charged to every other transfer, which the DFSP can choose to pass on

to the customer or not.

# Scenario Tests

## Setup

See [Jmeter setup here](#)

## Test Configuration

This is a functional end-to-end test and is driven by several components - environment configuration - test case data

### Environment Configuration:

The main configuration is external to the test in a csv file called **scenarioConfigData.csv**. This file should be in the same directory as the jmx file for the test. It contains the: *the name of the two DFSP servers - these can be made the same server* the USSD port number - it assumes that both servers use the same port number \* the expected total fee - it assumes the fee is the same in both directions These values can be adapted to your local environment by changing the file.

## Test Case Data

There are two versions of the test. In one version, **Scenario Test (new users).jmx**, two accounts are created dynamically and used for a transfer and an invoice. The user created will create users with a name in the format: TestUser\_DDDDDDD where the D's are digits. Those same digits are used for the user's national ID and PIN. The user's first name is always John. That version of the test writes two files: *ScenarioUser1.csv ScenarioUser2.csv*

These files contain the user digits, and the user number generated by the system.

In the second version, **Scenario Test (existing users from file).jmx**, the user csv files are used to send money and invoices between the existing users without creating new users or accounts.

## Test Functionality

The tests perform these scenarios: *Create new users on each DFSP* Check the initial balance (the amount is stored, but not validated) *Send Money from User1 to User2* Check the final balances and verify they are correct including the fee

*Send Invoice from User1 to User2* User2 approves the invoice, sending money back to User1 \* Balances are checked

Scenarios Not tested: account management, sub-accounts

## Logging and Output

- scenarioTest.log - produced by the test. Shows success/failure by step and performance info
- jmeter.log - produced by jmeter. Shows test failure info.

Tests can be modified to send results in mail.

# Integration Tests

Contains an initial attempt at providing performance and functional tests for Mojaloop assets.

## Setup

You will need [Apache JMeter 3.X](#) to load these files.

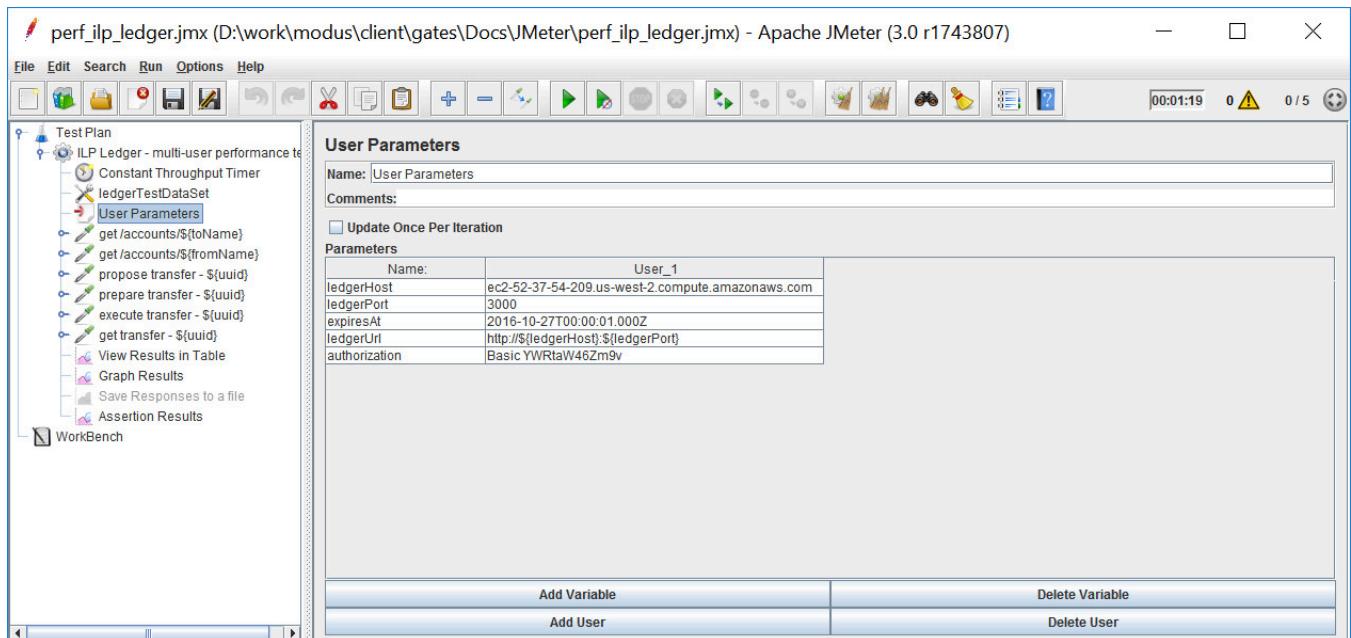
1. Download the appropriate archive for your operating system and expand it.
2. Download [jmeter-plugin-manager](#) and place it in the lib/ext folder under the path you extracted jmeter
3. Navigate to the /bin folder under the path you extracted jmeter
4. Start JMeter by executing jmeter.bat or jmeter.sh depending on your os
5. Navigate to Options Menu -> Plugin Manager -> Available Plugins (middle tab)
6. Select JMeter Plugins JSON and then click Apply Changes and Restart JMeter

## Test Configuration

The test is driven by several components - environment configuration - test case data - desired throughput - number of concurrent clients

### Environment Configuration:

User variables required to configure the test are located in the User Parameters element as shown below



In this section you should configure the following attributes:

attribute	description
ledgerHost	the host name the test should connect to
ledgerPort	the port
expiresAt	what data for expressAt should be included. Some date in the future.
ledgerUrl	by default its being built by host and port but you can change it here. these goes inside some of the request messages
authorization	the value to be included in the Authorization header element. This value is correct for admin / foo. This is a temporary hack.

## Test Case Configuration:

This test loads test cases from a csv file. You will need to either copy the sample file located here into your `jmeter/bin` directory or edit this element and include the path to the file. The file contains the following data.

The screenshot shows the Apache JMeter interface with a test plan titled "ILP Ledger - multi-user performance test". The left pane shows a tree view of the test plan, including a "Constant Throughput Timer" and a "CSV Data Set Config" element named "ledgerTestDataSet". The right pane is titled "CSV Data Set Config" and contains the following configuration:

- Name:** ledgerTestDataSet
- Comments:** (empty)
- Configure the CSV Data Source**
  - Filename:** perf\_ilp\_ledger\_testcases.csv
  - File encoding:** (empty)
  - Variable Names (comma-delimited):** fromName,toName,transferAmount
  - Delimiter (use "t" for tab):** ,
  - Allow quoted data?:** False
  - Recycle on EOF ?:** True
  - Stop thread on EOF ?:** False
  - Sharing mode:** All threads

fromAccountName, toAccountName, transferAmount

example:

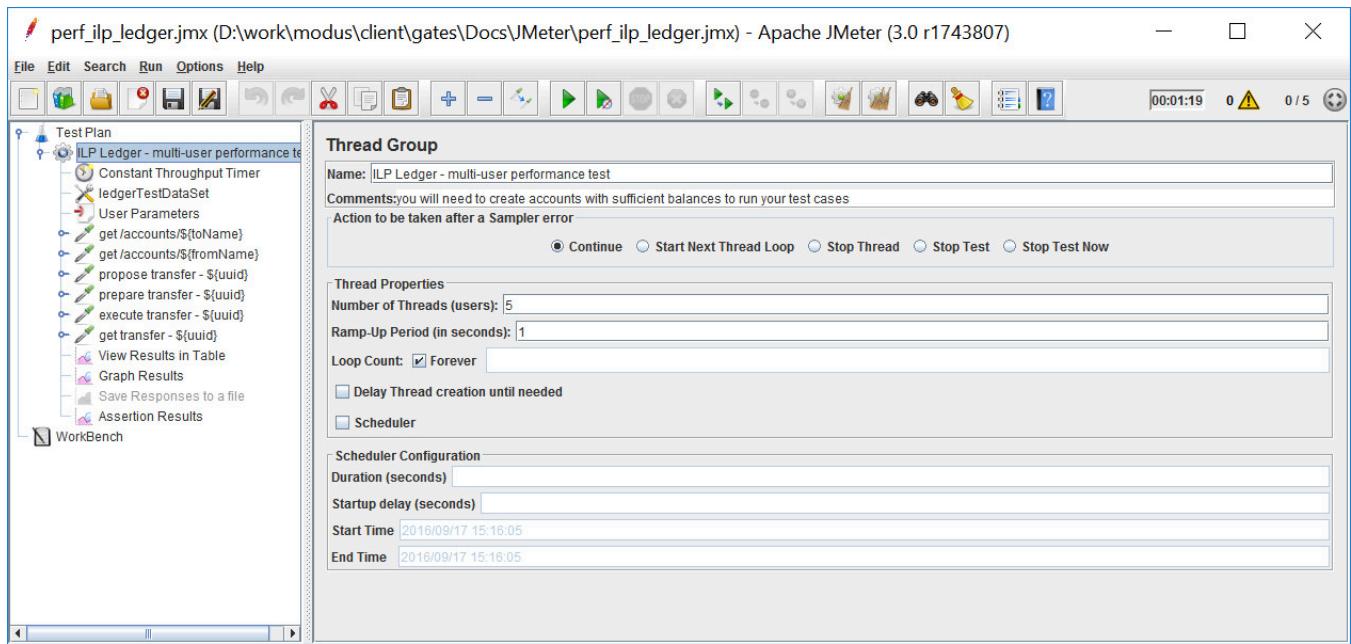
```
alice,bob,50
bob,alice,100
```

Each line in the file will cause be run by a client thread. When all of the lines have been executed the test will loop back to the beginning. This behavior can be changed.

Note that since this is a test that is intended to be run multiple times it does not create the accounts first. You will need to create accounts in your ledger with sufficient balances to sustain the test. When there are insufficient funds in a ledger you will start seeing errors pop up in the results table.

## Number of Concurrent Clients:

A `thread group` configures how many client threads should be used, how long they should take to ramp up to full load, and how many iterations of the test they should execute.



In this example we have 5 client threads ramping up over a 1 second window. They will loop forever. If the test receives an error it will continue to run.

As the number of client threads are increased throughput will increase to a point. There will be a point where throughput will continue to increase but per call latency will also start to climb. This is generally around the point where you cross over the number of threads allocated to the receiving thread pool on the service you are calling against. For example the default number of threads allocated to a http listener in mule is 16. As we increase client threads up to 16 throughput will steadily increase. Past 16 latency starts to climb because each client thread has to wait for some number of server side threads to complete before its request can be serviced. This can be good to a point because we are optimizing out part of the round trip latency between the computer making the request and the server servicing it. The time for the request to get there becomes free from the perspective of overall throughput.

## Throughput Configuration

The `constant throughput timer` constrains the amount of load generated by the client threads on the target system. It will constrain client connections so that they do not exceed the specified number of requests per minute. Note that these are total requests generated. The test case contains several requests per test iteration.

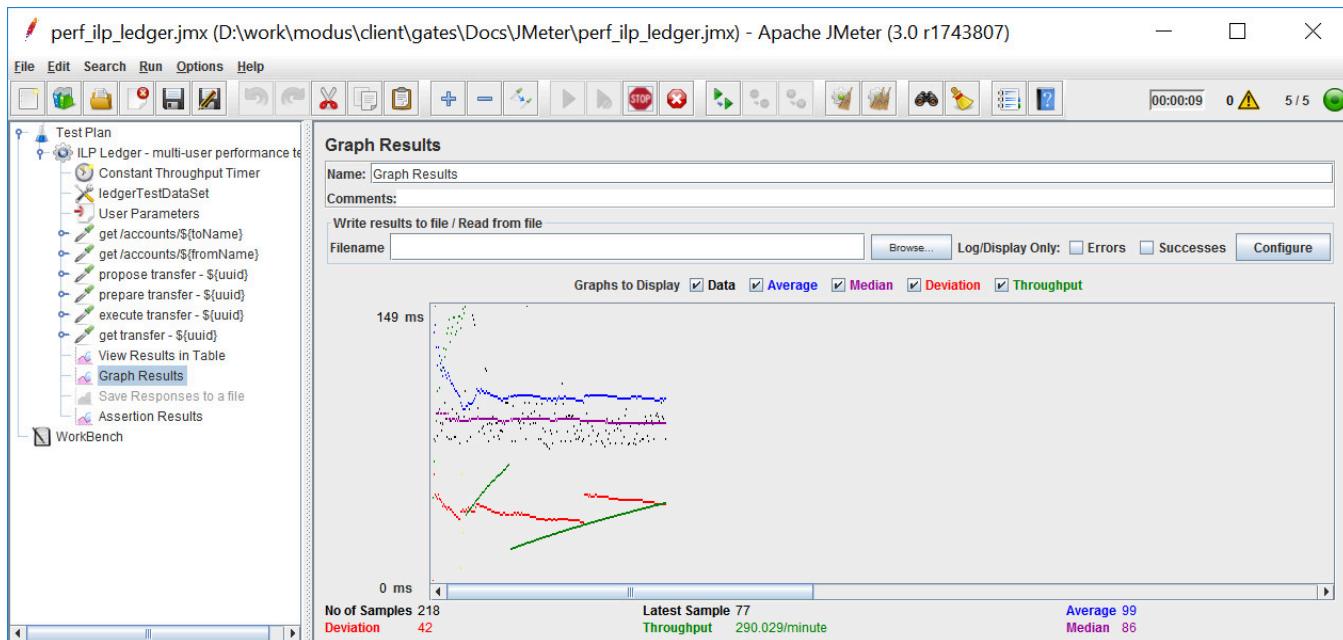
## Executing the Test

To start the test select Run → Start or click the button in the center of the toolbar. To stop the test select Run → stop or select the button. To clear results from previous test runs select the icon.

## Viewing Results

Results are being collected into a table and a graph. Other listeners can be configured.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency	Connect Time(m...)
1	13:28:29.835	ILP Ledger - mul...	get/accounts/bob	193	✓	510	193	117
2	13:28:30.029	ILP Ledger - mul...	get/accounts/all...	77	✓	513	77	0
3	13:28:30.033	ILP Ledger - mul...	get/accounts/bob	149	✓	510	149	68
4	13:28:30.148	ILP Ledger - mul...	propose transfer...	91	✓	885	91	0
5	13:28:30.182	ILP Ledger - mul...	get/accounts/all...	79	✓	513	79	0
6	13:28:30.234	ILP Ledger - mul...	get/accounts/all...	176	✓	513	176	85
7	13:28:30.329	ILP Ledger - mul...	prepare transfer...	93	✓	939	93	0
8	13:28:30.362	ILP Ledger - mul...	propose transfer...	86	✓	887	86	0
9	13:28:30.414	ILP Ledger - mul...	get/accounts/bob	82	✓	510	82	0
10	13:28:30.509	ILP Ledger - mul...	execute transfer ...	85	✓	278	85	0
11	13:28:30.435	ILP Ledger - mul...	get/accounts/bob	166	✓	510	166	84
12	13:28:30.603	ILP Ledger - mul...	prepare transfer...	97	✓	941	97	0
13	13:28:30.655	ILP Ledger - mul...	propose transfer...	88	✓	885	88	0
14	13:28:30.674	ILP Ledger - mul...	get/accounts/all...	78	✓	513	78	0
15	13:28:30.636	ILP Ledger - mul...	get/accounts/all...	160	✓	513	160	83
16	13:28:30.748	ILP Ledger - mul...	get transfer - 31...	77	✓	980	77	0
17	13:28:30.902	ILP Ledger - mul...	execute transfer ...	87	✓	278	87	0
18	13:28:30.936	ILP Ledger - mul...	get/accounts/bob	77	✓	510	77	0
19	13:28:30.955	ILP Ledger - mul...	prepare transfer...	93	✓	939	93	0
20	13:28:30.975	ILP Ledger - mul...	nontransf...	88	✗	987	88	0



## Debugging

If you are getting nothing but errors you can right click on the "Save Responses to a file" element near the bottom and choose enable. It will save the response from each step into its own file. You can also click on the Test Plan element at the top and select Functional Test Mode. This will cause JMeter to store the results of each call into a file. When debugging I recommend going into the Thread Group configuration and setting client threads to 1, unchecking forever and selecting 1 for the number of test iterations to execute. For this test the thread group element is the second one from the top and is called ILP Ledger - multi-user performance test

## Extending / Reusing

These basic tests can be used as the basis for testing any of our other rest based api's. There are many other connectors available other than rest as well.

### JSON Path Extractor:

This element shows extracting data from the response of a call to lookup an account. The `id` field contains the full path of the ledger account and is used in the subsequent call to construct the transfer request. The data is coming from response text, from the element named `id` in the root of the response object, and is being placed in a variable called `fromLedgerPath`. This will be referenced later is  `${fromLedgerPath}`

### JSON Path Assertion:

This element shows reading in the value `state` from the root of the response payload and asserting that it equals `executed`. The response code assertion further down checks to see if the status was 200.

# Resilience Modeling and Analysis

Wikipedia: "Failure mode and effects analysis (FMEA) . . . was one of the first systematic techniques for failure analysis. "

When FMEA is applied to software services it is called **Resilience Modeling and Analysis** (RMA), see white paper.

It was developed by reliability engineers in the late 1950s to study problems that might arise from malfunctions of military systems.

FMEA can be applied directly to software services to identify and rank possible service failures. Once identified, there are standard mitigation and testing patterns can be applied to resolve each failure. This process is used to prevent major failures and reduce downtime.

## What RMA is

Resilience modeling assumes that there will be failures in a system. It doesn't focus on increasing reliability, which is measured below as mean time to failure (MTTF), instead it focuses on reducing time to detection and recovery (MTTD and MTTR). By reducing time to detection and time to recovery (the red area below), availability (the green area) is maximized.

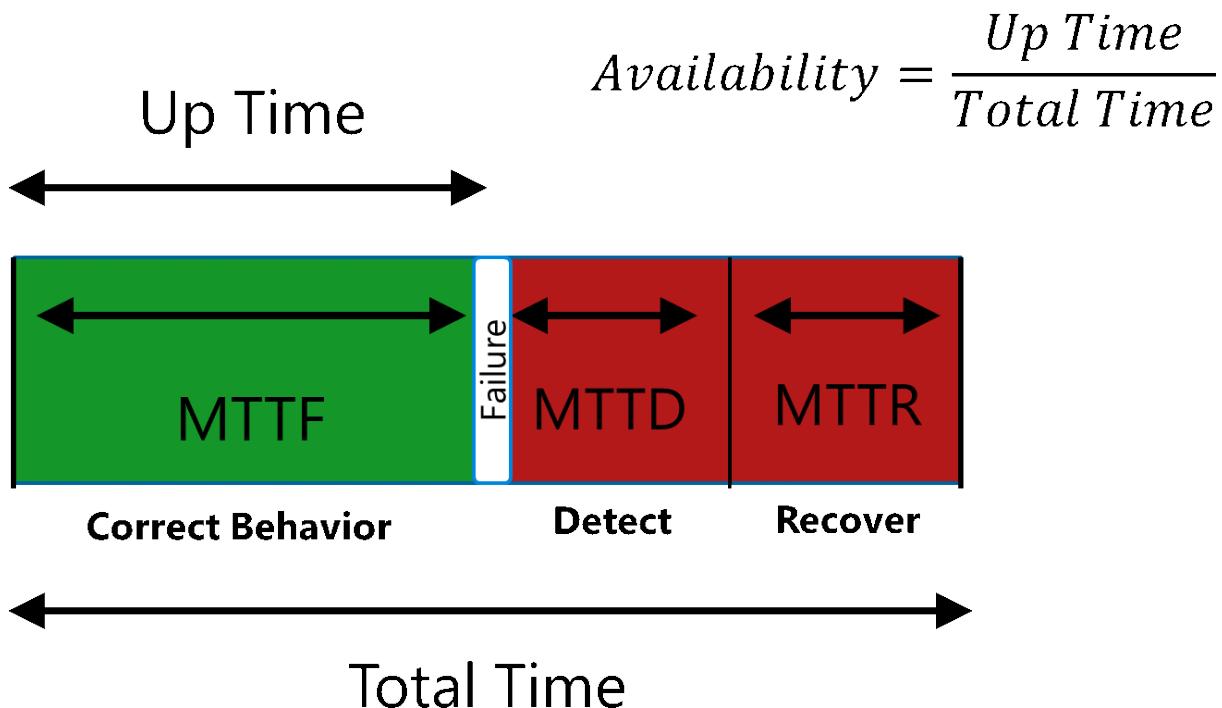


Figure - Availability from MTTF, MTTD, and MTTR

For RMA, developers look at the architecture data flows and ask: "what could go wrong here, how bad will that be, and how often will that happen?" RMA provides a prioritized list of potential faults. We extend RMA to add one or more ways to detect, mitigate, and test the handling for each of those faults.

RMA is a very similar process to threat modeling except that instead of looking for threats, we look for faults and instead of using a threat acronym like STRIDE we use DIAL:

- **D** - Discovery: name resolution, configuration
- **I** - Incorrectness: corruption, version mismatch, sequence errors, duplicates
- **A** - failure of Authorization or Authentication
- **L** - Latency; slow or no response, flooding, deadlocks, metering, timeouts

# Standard Microservice Resilience Patterns

Because Mojaloop follows a microservices architecture there are a group of standard potential failures that all such services have. Because every microservice has the same issues, these issues can be grouped together by failure mode along with standard methods of detection, mitigation, and testing.

When we apply DIAL to microservices, we find several standard patterns for failure.

## Failure Pattern #1 - Low Resources

A low resource condition is common to all software. It has the advantage that you can often detect and correct the problem before the failure occurs.

Example failures:

- Low memory
- Low disk space
- Excessive CPU
- Peak network traffic

### Detection

Use a system monitoring tool (Ex: AWS, Nagios, App Analytics, Sensu, New Relic, SCOM, etc.)

Two stages:

1. Yellow: Raise event when resource is getting low and before it's a problem
2. Red: Alert when the resource is critically low or gone

For each microservice we create a table. Here's an example:

Resource	Green	Yellow	Red
CPU	<80%	>80%, 1-minute average	> 95%, 10-minute average
Disk	<80% full	80 to 95% full	> 95%
Memory	<80% available memory utilized	80 to 95%, 3-minute average	> 95%, 5-minute average
Network	<80% network, capacity 5-minute average	80 to 95%, 5-minute average	> 95%

In Mojaloop, we make use of the ELK stack and Metricbeats for gathering system data. This makes the data available to any number of alerting systems.

### Mitigation

Graceful degradation: system continues to function, but some functionality may temporarily stop. As an example, in our case, new money transfer prepare requests might be slow or rejected while the services processes existing fulfillment work.

### Fault Injections

There are many standard tools to fill disk space, allocate large amounts of memory, hog CPU cycles, and throttle the network.

# Failure Pattern #2 - Service is down

Example Failures:

- Microservice down
- Mule down
- DB/SQL down

### Detection

In our case, each microservice implements a health endpoint which returns an http 200 if the service is up. Microservice may implement a JSON return to indicate that the service is degraded (yellow/warning status). The health service works by doing a simple internal check of the service.

We use the free Mule runtime which can be extended on-site for a license fee for monitoring and dashboards to cover this or another monitoring service may be used.

Service failures also are detected by a calling service when that service receives a connection failure (ex: 404). This works for dependent services that don't implement a health service. These failures are logged and picked up by the logging engine (ELK stack) where they can be integrated with a monitoring service.

## Mitigations

If service down is detected, the first mitigation is to restart the service. We use ansible playbooks for service startup.

Restart the service in a known good configuration. We use Ansible playbooks to deploy and configure the services. These can be run to redeploy and restart all the stateless services or restart the stateful ones. Restarting the service should generate a configuration change event.

Additional failover processes may be deployed in production.

The monitoring service should alert an operator when the service has been down for a threshold amount of time.

## Fault Injections

Stop the service

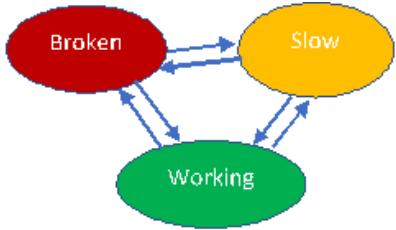
# Failure Pattern #3 - Health Modeling

## What is Health Modeling?

Health modeling is included here as a part of resilience, but it has a larger role in helping operations maintain the service. Health modeling answers. "what state is the service in, and what action can I take to correct it". In services, a health model defines a pattern that typically looks like "check the service state, attempt to fix it automatically if it's broken, alert the operator if we can't"

The first part of health modeling is defining the actionable states of the system.

A very simple health model might have three states: broken, slow, and working. The most general form of this model is a finite state model or petri net showing the three states and all possible transitions between them:



where the transitions are typically events that come from log events or health checks. Ex: The transition from Working to Broken might be "health check doesn't return 200".

In a simple model like this one, where the severity of the problems can be stack ranked, it's easy to model the system as a chain of responsibility pattern:

```

If (health state doesn't return 200) then broken

if (health state returns 200 with "slow" in JSON) then slow

else working

```

This kind of pattern is very easy to code and test. You can have any number of if/then statements in this kind of model, and multiple consecutive statements can lead to the same state. State checks are ordered from worst outcome to best. If any statement is true, the chain stops.

The second part of the health model is recovery process. This is also an ordered set of operations that can be shown as one or more chain of responsibility patterns - either for the entire system or for a group of states within it. Example:

```

if (broken for more than 5 minutes) alert operator

If (broken for more than 2 minutes) then raise event and run Ansible
playbook to redeploy service

If (broken) then raise event and run Ansible playbook to restart
service

```

```
if (slow and # of services > N) then alert operator

if (slow) then raise event and run playbook to add additional
microservice

if (working and more than 1 service and a service is idle) then run
playbook to scale down services
```

Describing the actions like this makes it easy to automate the responses and understand what should happen when problems occur.

# A General Microservice Health Model

An advantage of microservices is that every microservice has the same kinds of possible states and transitions. A general health model can apply to most of the operations in any microservice. Once we have that model we only need to worry about special cases specific to our service.

Our simple health model has only four states (in order):

- **Stopped** - the service is stopped or unresponsive
- **Misconfigured** - a catch-all state for something has gone wrong that the code can't automatically fix. We don't support auto-rollback of a new deployment, but If you want to you can add "Mis-deployed" state above this one to cover the case where a new deployment has been done N minutes ago yet is still in the misconfigured state.
- **Slow** - performance is below an acceptable threshold
- **Working**

The main difference between the simple health model example above and our model is the addition of the misconfigured state. Below are details for handling each state.

## Stopped Service

Example Failures:

- Microservice down
- Mule down
- DB/SQL down

## Detection

In our case, each microservice implements a health endpoint which returns an http 200 if the service is up. Microservice may implement a JSON return to indicate that the service is degraded (yellow/warning status). The health service works by doing a simple internal check of the service. We use the free Mule runtime which can be extended on-site for a license fee for monitoring and dashboards to cover this or another monitoring service may be used.

Service failures also are detected by a calling service when it receives a connection failure (ex: 404). This works for dependent services that don't implement a health service. These failures are logged and picked up by the logging engine (ELK stack) where they can be integrated with a monitoring service.

## Mitigations

```
If the service has been down for N + M minutes alert an operator
Else, restart the service using an ansible playbook
```

Additional failover processes may be deployed in production. Restarting the service generates a configuration change event.

## Fault Injection

Stop the service to test that the service will be restarted.

## Configuration error examples

Many possible failures lead to the misconfigured state. In all cases, the configuration error detection can come from a logged message since the service is running and logging, it's just not communicating. That message should have a log type to indicate that there's a config error. The type of checks will depend on what communication methods the microservices supports. Here's a list:

### Http

- Auth: major security failure. Unable to call upstream service and/or all clients can't get data
- Misconfigured URL
- Misconfigured network
- API version mismatch

## Web sockets

- Multiple clients on same socket
- Port not open or configured (ex: in Docker)
- Socket not configured (DFSP initiates)
- Major version mismatch
- client access auth failure - client service logs config error
- error on notify - receiver logs error (we may consider retries here before failing)

## SQL

- Misconfigured Connection string
- Misconfigured network

**General** - infrastructure version incorrect (ex: OS, Docker, JScript)

Besides all the config failures, it can be helpful if the service has a "configuration good" log event that gets fired after startup or a configuration change. This allows the model to know when a state has returned to "working". Since we deal with configuration problems manually, this is not required, but in an automated setup it would be needed.

## Detection

- Run a scheduled test to ping the health service
- Listen for actionable log messages marked with the configuration type

## Mitigation

Use the mitigations above for restarting the service, but add this check in the middle:

- If the service has been down for N minutes. Use an Ansible playbook to redeploy and configure the service. A stateful service can be also reinstalled, but leaving the existing data volume untouched.

## Fault Injections

- Change Http or socket config

- Change client or server auth
- Force update of component to incorrect version or configuration

## Slow Service

If performance is below an acceptable threshold the health state will return that. The model here follows the example above

### Detection

- if (health state returns 200 with "slow" in JSON) then slow

### Mitigation

- if (slow) then raise event and run playbook to add additional microservice
- if (working and more than 1 service and a service is idle) then run playbook to scale down

### Fault Injection

- Use a tool to load the processor

# Mojaloop Specific Health Modeling

Mojaloop has two additional potential faults that need to be addressed that could cause a participating DFSP to lose money. These are *overloaded ledgers* and *dropped messages*.

# Overloaded Ledger

Example: Payer sends \\$100. Payee DFSP agrees and starts fulfilment. The payer ledger is overload and doesn't resolve the transfer in time, however, it's been fulfilled by the payee DFSP and the center. Payer DFSP losses \\$100 during settlement.

A similar problem happens if the central ledger doesn't resolve the transfer. Then the payee DFSP can be out the \\$100 during settlement.

## Detection

Track remaining time on all transfers. If a transfer is not reported before the timeout window (either fulfilled, rejected, or cancelled), then it's delinquent and needs to be checked on.

## Mitigation

1) Thread priority: Ledgers handle fulfillments before preparing new transfers. 2) Query on timeout: If payer DFSP hasn't explicitly heard a "fulfil", "reject", or "cancel" message at the end of a transfer timeout, it queries the center to get the current status of the transfer. The payee DSFP can check the status at anytime, such as before a settlement window or on restart of its services after a failure.

Extending this pattern: This solution treats the central ledger as the source of truth. It can be extended to multiple hubs where the transfer goes through many hops before getting to the final destination. This works following an eventual consistency model the same as above but with multiple central hubs. Each central hub acts exactly the same way and uses the same transfer ID for the transfer. At the end of a timeout, if any participant doesn't know the status, they check and retry the next participant up the chain. Whenever a participant changes a ledger they send status both up and back down the chain. On the downside, they should get a corresponding change notification. If that doesn't happen they retry.

## Fault Injection

Take down the payee ledger adapter

# Messages dropped

As with the ledger overload problem, if the ledger notifications are missed or dropped the payer or payee ledger can lose money.

**Detection** 1. Payer DFSP detects when no message is received within the timeout 2. Payee DFSP doesn't receive an ACKnowledge and fulfillment notification from the central ledger.

**Mitigation** 1) Query on timeout: The payer DFSP can resolve this with the same mechanism as an overloaded ledger.  
2) Wait for Notify Fulfillment message: The payee DFSP would lose money if it fulfills a transfer, but doesn't deliver the notification of it to the central ledger. To mitigate this, the payee DFSP doesn't notify the payee or hand any money out until it receives an fulfillment notification from the central ledger. The payee DFSP can choose to check on a transfer status with the center, but it is not recommended to do this for every transfer. 3) Retries (w/idempotent writes): the ILP-Connector may retry fulfillment messages automatically (opt-in) if there's no response. For this to work properly, the ledgers must implement idempotent writes on the transfer ID.

## Fault Injection

Drop/block the fulfillment notifications

# PKI Guide

---

## Introduction

In this guide, we will introduce some features of CloudFlare PKI -- [cfssl](#) CFSSL is a tool developed by CloudFlare. It's both a command line tool and an HTTP API server for signing, verifying and bundling TLS certificates. It requires GO 1.6+ to build. In this guide, we use the command line tool as the example.

- [Background](#)
  - [Rationale](#)
  - [Install cfssl](#)
  - [CA Config](#)
  - [Client Config](#)
  - [Key Suggestions](#)
  - [Integrating the Certificates with Service](#)
- 

## Background

Secure Channels enable confidentiality and integrity of data across network connections. In the context of Mojaloop, a secure channel can be made possible by the implementation of service transport security via TLS to protect data in-transit and enable mutual authentication. The centralization of trust in a TLS implementation is provided through a Public Key Infrastructure (PKI).

**Note:** While the Central KMS may serve as a PKI as the Central Services evolve, an existing internal or hosted PKI can provide the management and distribution of certificates for service endpoints.

TLS helps mitigate a number of identified threats discovered during Mojaloop Threat Modeling exercises:

- **Tampering:** Network traffic sniffing and or manipulation across DFSP, Pathfinder and Central Services
- **Spoofing:**
  1. Rogue DFSP pretends to be another DFSP at central directory
  2. False connector subscribes to notifications for transfers
  3. Notifications are sent by a party other than the central ledger
  4. Rogue KMS requests a health check or log inquiry to Forensic Logging Sidecars
  5. Data manipulation of REST calls
- **Information Disclosure:**
  1. A false connector or 3rd party connector subscribes to notifications that are not theirs
  2. Inappropriate use of Cryptography (including side-channel weaknesses)
- **Elevation of Privilege:**
  1. Credential Exposure by DFSP
  2. Credential Exposure by Customer
  3. Credential Exposure by Central Services Employee

## Rationale

The implementation of TLS is a deployment-specific consideration as the standards, configurations and reliance on a PKI are best defined by the implementor. Mojaloop team has demonstrated a PKI/TLS design which may be configured and implemented to meet the needs of a deployment scenario through the use of the CloudFlare PKI Toolkit.

This toolkit provides a central root of trust, an API for automation of certificate activities and configuration options which optimize the selection of safe choices while abstracting low-level details such as the selection and implementation of low-level cryptographic primitives. An introduction to this toolkit with safe examples for the generation and testing of certificates is found below.

## Install cfssl

To install the cfssl tool, please follow the instructions for Cloudflare's [cfssl](#)

## CA Config

### Initialize a certificate authority

First, you need to configure the certificate signing request (csr), which we've named `ca.json`. For the key algorithm, rsa and ecdsa are supported by cfssl, but you need to avoid using a small sized key.

```
{
 "hosts": [
 "root.com",
 "www.root.com"
],
 "key": {
 "algo": "rsa",
 "size": 2048
 },
 "names": [
 {
 "C": "US",
 "L": "Des Moines",
 "O": "Dwolla, Inc.",
 "OU": "Mojaloop",
 "ST": "Iowa"
 }
]
}
```

```

Then you need to generate a cert and the related private key for the CA.

```
```
cfssl gencert -initca ca.json | cfssljson -bare ca -
```

```

You will receive the following files:

`ca-key.pem` `ca.csr` `ca.pem`

```
* `ca.pem` is your cert.  
* `ca-key.pem` is your related private key, which should be stored  
in a safe spot. It will allow you to sign any cert.
```

```
#### Run a CA server
```

To run a CA server, you need the ``ca-key.pem`` and ``ca.pem`` files from the first step, and a config file, ``config_ca.json``, for the server.

```
{ "signing": { "default": { "auth_key": "central_ledger", "expiry": "8760h", "usages": [ "signing", "key encipherment", "server auth", "client auth" ], "name_whitelist": "\.central-ledger\.com\$" } }, "auth_keys": { "central_ledger": { "key": "0123456789abcdef", "type": "standard" } } }
```

- `auth_key` is the token used to authenticate the client's CSR.
- `expiry` is the valid time period for the cert. A year is around 8760 hours.
- `name_whitelist` is the regular expression for the domain names that can be signed by the CA. To run the server:

```
cfssl serve -ca=ca.pem -ca-key=ca-key.pem -config=config_ca.json  
-port=6666
```

The default IP and port number is: 127.0.0.1:8888.

Client Config

To generate a certificate for the client, you will need a config file -- `config_clients.json` for cfssl.

```
{
  "auth_keys" : {
    "central_ledger" : {
      "type" : "standard",
      "key" : "0123456789abcdef"
    }
  },
  "signing" : {
    "default" : {
      "auth_remote" : {
        "remote" : "ca_server",
        "auth_key" : "central_ledger"
      }
    }
  },
}
```

```

    "remotes" : {
        "ca_server" : "localhost:6666"
    }
}
```
* The authentication token in `auth_keys` must match the one in the server.
* The server address in `remotes` must match the real server address. You will also need another config file -- `central_ledger.json` for the service.

```

```
{ "hosts": ["www.central-ledger.com"], "key": { "algo": "ecdsa", "size": 256 }, "names": [{ "C": "US", "L": "Des Moines", "O": "Mojaloop", "OU": "leveloneproject-central-services", "ST": "Iowa" }] } ``
```

- The domain name in hosts must match the whitelist in config\_ca.json.
- You should avoid using a small sized key in key. To generate a certificate for the service:

```
cfssl gencert -config=config_clients.json central_ledger.json | cfssljson -bare central_ledger
```

You will receive the following files:

```
central_ledger-key.pem
central_ledger.csr
central_ledger.pem
```

central\_ledger.pem will be your service's cert, and central\_ledger-key.pem will be your private key.

## Key Suggestions

During the certificate signing requests, we suggest you avoid using small keys. The minimum requirement is shown in the following table:

Signature Key	RSA	ECC
AES-256	>=2048	PCurves >= 256

## Integrating the certificates with service

Once the certificates have been created, you will need to integrate them with your service. We will use central-ledger as

an example here.

## Server

On the server side, you'll want to set it up so that every incoming request is checked against the cert. In the projects so far, our team has used Hapi, but it's just as simple when using Node libraries. Both methods are shown to make the process as straightforward as possible.

### Setting up with Hapi

On the server side, we simply added the key and cert to a tls object. When the server connection is initialized, the tls object is added as an option. The fs library is used to read the key and cert files.

```
var fs = require('fs')
.

.

.

const tls = {
 key: fs.readFileSync('./src/ssl/central_ledger-key.pem'),
 cert: fs.readFileSync('./src/ssl/central_ledger.pem')
}
const server = new Hapi.Server()
server.connection({
 tls
})
```

### Setting up with Node

This step doesn't change much. The key and cert along with the client cert are added to an options object. When the server is created, they are added as options. The fs library is used to read the key and cert files.

```
var fs = require('fs')
var https = require('https')

.

.

.

var options = {
 key: fs.readFileSync('central_ledger-key.pem'),
 cert: fs.readFileSync('central_ledger.pem'),
 ca: fs.readFileSync('ca.pem'),
}

https.createServer(options, function (req, res) {
 res.writeHead(200)
```

```
) .listen(3000)
```

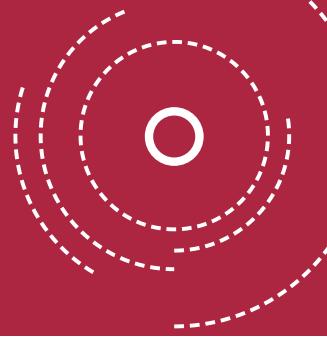
## Client

For client requests to the server, we use many of the same libraries. fs is used to read the client cert and https is used to make the requests. The cert simply needs to be added as part of the options object under the name ca.

```
var fs = require('fs')
var https = require('https')
var options = {
 hostname: 'localhost',
 port: 3000,
 path: '/',
 method: 'GET',
 ca: fs.readFileSync('ca.pem')
}

var req = https.request(options, function(res) {
 res.on('data', function(data) {
 })
})
req.end()
```

# Appendix



## Terminology

These are the preferred terms and definitions for Mojaloop.

## Details

Term	Alternative and Related Terms	Mojaloop Definition
Access Point	POS ("Point of Sale"), Customer Access Point, ATM, Branch	Places or capabilities that are used to initiate or receive a payment. Access points can include bank branch offices, ATMs, terminals at the POS, agent outlets, mobile phones, and computers.
Account Lookup System		Account Lookup System is an abstract entity used for retrieving information regarding in which FSP an account, wallet or identity is hosted. The Account Lookup System itself can be hosted in its own server, as part of a financial switch, or in FSPs.
Account Number	Account ID	A unique number representing an account. There can be multiple accounts for each end user.
Active User		A term used by many providers in describing how many of their account holders are frequent users of their service.
Addressing	Directories, Aliasing	The use of necessary information (account number, phone number, etc.) for a paying user to direct payment to a receiving user.
Agent	Agent till, Agent outlet	An entity authorized by the provider to handle various functions such as customer enrollment, cash-in and cash-out using an agent till.
Agent Outlet	Access point	A physical location that carries one or more agent tills, enabling it to perform

Term	Alternative and Related Terms	Mojaloop Definition
		enrollment, cash-in and cash-out transactions for customers on behalf of one or more providers. National law defines whether an agent outlet may remain exclusive to one provider. Agent outlets may have other businesses and support functions.
Agent Till	Registered agent	An agent till is a provider-issued registered "line", either a special SIM card or a POS machine, used to perform enrollment, cash-in and cash-out transactions for clients. National law dictates which financial service providers can issue agent tills.
Aggregator	Merchant Aggregator	A specialized form of a merchant services provider who typically handles payments transactions for a large number of small merchants. Scheme rules often specify what aggregators are allowed to do.
Anti Money Laundering	AML; also "Combating the Financing of Terrorism", or CFT	Initiatives to detect and stop the use of financial systems to disguise use of funds criminally obtained.
Application Program Interface	API	A software program that makes it possible for application programs to interact with each other and share data.
Arbitration		The use of an arbitrator, rather than courts, to resolve disputes.
Authentication	Verification, Validation	The process of ensuring that a person or a transaction is valid for the process (account opening, transaction initiation, etc.) being performed.
Authorization		A process used during a "pull" payment (such as a card payment), when the payee requests (through their provider) confirmation from the payer's bank that the transaction is good.
Automated Clearing House		An electronic clearing system in which payment orders are exchanged among payment service providers, primarily via magnetic media or telecommunications networks, and then cleared amongst the participants. All operations are handled by a data processing center. An ACH typically clears credit transfers and debit transfers, and in some cases also cheques.
Bank	Savings Bank, Credit Union, Payments Bank	A charted financial system within a country that has the ability to accept deposits and make and receive payments into those accounts.
Bank Accounts and Transaction Services	Mobile Banking, Remote Banking, Digital	A transaction account held at a bank. This account may be accessible by a mobile phone, in which case it is sometimes referred to as "mobile banking".

Term	Alternative and Related Terms	Mojaloop Definition
	Banking	
Bank-Led Model	Bank-Centric Model	A reference to a system in which banks are the primary providers of digital financial services to end users. National law may require this.
Basic Phone		Minimum device required for DFS
Bilateral Net Settlement System		A settlement system in which participants' bilateral net settlement positions are settled between every bilateral combination of participants.
Bilateral Netting		An arrangement between two parties to net their bilateral obligations. The obligations covered by the arrangement may arise from financial contracts, transfers or both.
Bill Payment	C2B, Utility payments, school payments	Making a payment for a recurring service, either in person ("face to face") or remotely.
Biometric Authentication		The use of a physical characteristic of a person (fingerprint, IRIS, etc.) to authenticate that person.
Blacklist		A list or register of entities (registered users) that are being denied/blocked from a particular privilege, service, mobility, access or recognition. Entities on the list will NOT be accepted, approved and/or recognized. It is the practice of identifying entities that are denied, unrecognized, or ostracized. Where entities are registered users (or user accounts, if granularity allows) and services are informational (e.g. balance check), transactional (e.g. debit/credit) payments services or lifecycle (e.g. registration, closure) services.
Blockchain	Digital currency, cryptocurrency, distributed ledger technology	The technology underlying bitcoin and other cryptocurrencies-a shared digital ledger, or a continually updated list of all transactions.
Borrowing		Borrowing money to finance a short term or long term need
Bulk Payer		An organization (or rarely, an individual), that needs to pay to many users at once.
Bulk Payments	G2C, B2C , G2P, social transfers	Making and receiving payments from a government to a consumer: benefits, cash transfers, salaries, pensions, etc.
Bulk Payments Services		A service that allows a government agency or an enterprise to make payments to a large number of payees - typically consumers but can be

Term	Alternative and Related Terms	Mojaloop Definition
Bulk upload service		businesses as well.
Bundling	Packaging, Tying	A service enabling the import of multiple transactions per session, most often via a bulk data transfer file which is used to initiate payments. Example: salary payment file.
Business		A business model in which a provider which groups a collection of services into one product which an end user agrees to buy or use.
Cash Management	Agent Liquidity Management	Entity such as a public limited or limited company or corporation that uses mobile money as a service, e.g. taking bill payments, making bill payments and disbursing salaries
Cash-In		Management of cash balances at an agent.
Cash-Out		Receiving eMoney credit in exchange for physical cash - typically done at an agent.
Chip Card	EMV Chip Card, Contactless Chip Card	Receiving physical cash in exchange for a debit to an eMoney account - typically done at an agent.
CICO		A chip card contains a computer chip: it may be either contactless or contact (requires insertion into terminal). Global standards for chip cards are set by EMV.
Clearing		Cash In Cash Out
Clearing House		The process of transmitting, reconciling, and, in some cases, confirming transactions prior to settlement, potentially including the netting of transactions and the establishment of final positions for settlement.
		Sometimes this term is also used (imprecisely) to cover settlement. For the clearing of futures and options, this term also refers to the daily balancing of profits and losses and the daily calculation of collateral requirements.
Closed-Loop		A central location or central processing mechanism through which financial institutions agree to exchange payment instructions or other financial obligations (e.g. securities). The institutions settle for items exchanged at a designated time based on the rules and procedures of the clearinghouse. In some cases, the clearinghouse may assume significant counterparty, financial, or risk management responsibilities for the clearing system.
Combatting	CFT (Counter	A payment system used by a single provider, or a very tightly constrained group of providers.
		Initiatives to detect and stop the use of financial systems to transfer funds to

Term	Alternative and Related Terms	Mojaloop Definition
Terrorist Financing	Financing of Terrorism)	terrorist organizations or people.
Commission		An incentive payment made, typically to an agent or other intermediary who acts on behalf of a DFS provider. Provides an incentive for agent.
Commit		Commit means that the electronic funds that were earlier reserved are now moved to the final state of the financial transaction. The financial transaction is completed. The electronic funds are no longer locked for usage.
Counterparty	Payee, payer, borrower, lender	The other side of a payment or credit transaction. A payee is the counterparty to a payer, and vice-versa.
Coupon		A token that entitles the holder to a discount or that may be exchanged for goods or services
Credit History	Credit bureaus, credit files	A set of records kept for an end user reflecting their use of credit, including borrowing and repayment.
Credit Risk Management		Tools to manage the risk that a borrower or counterparty will fail to meet its obligations in accordance with agreed terms.
Credit Scoring		A process which creates a numerical score reflecting credit worthiness.
Cross Border Trade Finance Services		Services which enable one business to sell or buy to businesses or individuals in other countries; may include management of payments transactions, data handling, and financing.
Cross-FX Transfer		Transfer involving multiple currencies including a foreign exchange calculation
Customer Database Management		The practices that providers do to manage customer data: this may be enabled by the payment platform the provider is using.
Data Protection	PCI-DSS	The practices that enterprises do to protect end user data. "PCI-DSS" is a card industry standard for this.
Deposit Guarantee System	Deposit Insurance	A fund that insures the deposits of account holders at a provider; often a government function used specifically for bank accounts.
DFSP On-boarding		On-boarding a DFSP is the process of adding a new DFSP to this financial network.
Digital Financial		The regulated entity providing digital financial services to users. Manages the wallet for the users. May manage other types of digital assets such as savings

Term	Alternative and Related Terms	Mojaloop Definition
Service Provider (DFSP)		accounts, loans etc. Depending on countries and regulations, a DFSP can be a bank, a telco, a Mobile Money Operator or some other private entity.
Digital Financial Services	Mobile Financial Services	Digital financial services include methods to electronically store and transfer funds; to make and receive payments; to borrow, save, insure and invest; and to manage a person's or enterprise's finances.
Digital Liquidity		A state in which a consumer willing to leave funds (eMoney or bank deposits) in electronic form, rather than performing a "cash-out".
Digital Payment	Mobile Payment, Electronic Funds Transfer	A broad term including any payment which is executed electronically. Includes payments which are initiated by mobile phone or computer. Card payments in some circumstances are considered to be digital payments. The term "mobile payment" is equally broad, and includes a wide variety of transaction types which in some way use a mobile phone.
Dispute Resolution		A process specified by a provider or by the rules of a payment scheme to resolve issues between end users and providers, or between an end user and its counter party.
Domestic Remittance	P2P; Remote Domestic Transfer of Value	Making and receiving payments to another person in the same country.
Electronic Invoicing, ERP, Digital Accounting, Supply Chain Solutions Services, Business Intelligence		Services that support merchant or business functions relating to DFS services.
eMoney	eFloat, Float, Mobile Money, Electronic Money, Prepaid Cards	A record of funds or value available to a consumer stored on a payment device such as chip, prepaid cards, mobile phones or on computer systems as a non-traditional account with a banking or non-banking entity.
eMoney Accounts and Transaction	Digital Wallet, Mobile Wallet, Mobile Money	A transaction account held at a non-bank. The value in such an account is referred to as eMoney.

Term	Alternative and Related Terms	Mojaloop Definition
Services	Account	
eMoney Issuer	Issuer, Provider	A provider (bank or non-bank) who deposits eMoney into an account they establish for an end user. eMoney can be created when the provider receives cash ("cash-in") from the end user (typically at an agent location) or when the provider receives a digital payment from another provider.
Encryption	Decryption	The process of encoding a message so that it can be read only by the sender and the intended recipient.
End User	Consumer, Customer, Merchant, Biller	The customer of a digital financial services provider: the customer may be a consumer, a merchant, a government, or another form of enterprise.
Escrow	Funds Isolation, Funds Safeguarding, Custodian Account, Trust Account.	A means of holding funds for the benefit of another party. eMoney Issuers are usually required by law to hold the value of end users' eMoney accounts at a bank, typically in a Trust Account. This accomplishes the goals of funds isolation and funds safeguarding.
External Account		An account hosted outside the FSP, regularly accessible by an external provider interface API.
FATF		The Financial Action Task Force is an intergovernmental organization to combat money laundering and to act on terrorism financing.
Feature Phone		A mobile telephone without significant computational capabilities.
Fees		The payments assessed by a provider to their end user. This may either be a fixed fee, a percent-of-value fee, or a mixture. A Merchant Discount Fee is a fee charged by a Merchant Services Provider to a merchant for payments acceptance. Payments systems or schemes, as well as processors, also charge fees to their customer (typically the provider.)
Financial Inclusion		The sustainable provision of affordable digital financial services that bring the poor into the formal economy.
Financial Literacy		Consumers and businesses having essential financial skills, such as preparing a family budget or an understanding of concepts such as the time value of money, the use of a DFS product or service, or the ability to apply for such a service.
FinTech		A term that refers to the companies providing software, services, and products for digital financial services: often used in reference to newer technologies.

Term	Alternative and Related Terms	Mojaloop Definition
Float		This term can mean a variety of different things. In banking, float is created when one party's account is debited or credited at a different time than the counterparty to the transaction. eMoney, as an obligation of a non-bank provider, is sometimes referred to as float.
Fraud	Fraud Management, Fraud Detection, Fraud Prevention	Criminal use of digital financial services to take funds from another individual or business, or to damage that party in some other way.
Fraud Risk Management	Also known as fraud risk management service (FRMS)	Tools to manage providers' risks, and at times user's risks (e.g. for merchants or governments) in providing and/or using DFS services.
FX		Foreign Exchange.
Government Payments Acceptance Services		Services which enable governments to collect taxes and fees from individuals and businesses.
HCE		A communication technology that enables payment data to be safely stored without using the Secure Element in the phone.
Identity	National Identity, Financial Identity, Digital Identity	A credential of some sort that identifies an end user. National identities are issued by national governments. In some countries a financial identity is issued by financial service providers.
Immediate Funds Transfer	Real Time	A digital payment which is received by the payee almost immediately upon the payer having initiated the transaction.
Insurance Products		A variety of products which allow end user to insure assets or lives that they wish to protect.
Insuring Lives or assets		Paying to protect the value of a life or an asset.
Interchange	Swipe Fee, Merchand Discount Fee	A structure within some payments schemes which requires one provider to pay the other provider a fee on certain transactions. Typically used in card schemes to effect payment of a fee from a merchant to a consumer's card issuing bank.

Term	Alternative and Related Terms	Mojaloop Definition
International Remittance	P2P; Remote Cross-border Transfer of Value, Cross-Border Remittance	Making and receiving payments to another person in another country.
Interoperability	Interconnectivity	When payment systems are interoperable, they allow two or more proprietary platforms or even different products to interact seamlessly. The result is the ability to exchange payments transactions between and among providers. This can be done by providers participating in a scheme, or by a variety of bilateral or multilateral arrangements. Both technical and business rules issues need to be resolved for interoperability to work.
Interoperability settlement bank		Entity that facilitates the exchange of funds between the FSPs. The settlement bank is one of the main entities involved in any inter-FSP transactions.
Investment Products		A variety of products which allow end users to put funds into investments other than a savings account.
Irrevocable	Non-Repudiation	A transaction that cannot be "called back" by the payer; an irrevocable payment, once received by a payee, cannot be taken back by the payer.
Interoperability service for transfer	IST	Inter system trunk that allows for routing of payments.
Know Your Customer	KYC, Agent and Customer Due Diligence, Tiered KYC, Zero Tier	The process of identifying a new customer at the time of account opening, in compliance with law and regulation. The identification requirements may be lower for low value accounts ("Tiered KYC"). The term is also used in connection with regulatory requirements for a provider to understand, on an ongoing basis, who their customer is and how they are using their account.
L1P Bulk Payment Facilitator		An organization that processes L1P compliant payments and resulting reports on behalf of Bulk Payers.
Liability	Agent Liability, Issuer Liability, Acquirer Liability	A legal obligation of one party to another; required by either national law, payment scheme rules, or specific agreements by providers. Some scheme rules transfer liabilities for a transaction from one provider to another under certain conditions.
Liquidity	Agent liquidity	The availability of liquid assets to support an obligation. Banks and non-bank providers need liquidity to meet their obligations. Agents need liquidity to

Term	Alternative and Related Terms	Mojaloop Definition
meet cash-out transactions by consumers and small merchants.		
Loans	Microfinance, P2P Lending, Factoring, Cash Advances, Credit, Overdraft, Facility	Means by which end users can borrow money.
M2C		Merchant to Customer or Consumer.
mCommerce	eCommerce	Refers to buying or selling in a remote fashion: by phone or tablet (mCommerce) or by computer (eCommerce)
Merchant	Payments Acceptor	An enterprise which sells goods or services and receives payments for such goods or services.
Merchant Acquisition	Onboarding	The process of enabling a merchant for the receipt of electronic payments.
Merchant payment - POS	C2B, Proximity Payments	Making a payment for a good or service in person ("face to face"); includes kiosks and vending machines.
Merchant payment - Remote	C2b, eCommerce Payment, Mobile Payment	Making a payment for a good or service remotely; transacting by phone, computer, etc.
Merchant Payments Acceptance Services	Acquiring services	A service which enables a merchant or other payment acceptor to accept one or more types of electronic payments. The term "acquiring" is typically used in the card payments systems.
Merchant Service Provider	Acquirer	A provider (bank or non-bank) who supports merchants or other payments acceptors requirements to receive payments from customers. The term "acquirer" is used specifically in connection with acceptance of card payments transactions.
MFSP Platform		Mobile financial service providers
Mobile Network Operator		An enterprise which sells mobile phone services, including voice and data communication.
Money Transfer Operator		A specialized provider of DFS who handles domestic and/or international remittances.

Term	Alternative and Related Terms	Mojaloop Definition
Multilateral Net Settlement Position		The sum of the value of all the transfers a participant in a net settlement system has received during a certain period of time less the value of the transfers made by the participant to all other participants. If the sum is positive, the participant is in a multilateral net credit position; if the sum is negative, the participant is in a multilateral net debit position.
Multilateral Net Settlement System		A settlement system in which each settling participant settles (typically by means of a single payment or receipt) the multilateral net settlement position which results from the transfers made and received by it, for its own account and on behalf of its customers or non-settling participants for which it is acting.
Multilateral Netting		Netting on a multilateral basis is arithmetically achieved by summing each participant's bilateral net positions with the other participants to arrive at a multilateral net position. Such netting is conducted through a central counterparty (such as a clearing house) that is legally substituted as the buyer to every seller and the seller to every buyer. The multilateral net position represents the bilateral net position between each participant and the central counterparty.
NDFSP		national digital financial service providers
Near Field Communication	NFC	A communication technology used within payments to transmit payment data from an NFC equipped mobile phone to a capable terminal.
Netting		The offsetting of obligations between or among participants in the settlement arrangement, thereby reducing the number and value of payments or deliveries needed to settle a set of transactions.
Non Bank-Led Model	MNO-Led Model	A reference to a system in which non-banks are the providers of digital financial services to end users. Non-banks typically need to meet criteria established by national law and enforced by regulators.
Non-Bank	Payments Institution, Alternative Lender	An entity that is not a chartered bank, but which is providing financial services to end users. The requirements of non-banks to do this, and the limitations of what they can do, are specified by national law.
Nostro Account		From the Payer's perspective: Payer FSP funds/accounts held/hosted at Payee FSP
Notification		Notice to payer or payee regarding the status of a transfer.
Off-Us Payments	Off-net payments	Payments made in a multiple-participant system or scheme, where the payer's provider is a different entity as the payee's provider.

Term	Alternative and Related Terms	Mojaloop Definition
On-Us Payments	On-net payments	Payments made in a multiple-participant system or scheme, where the payer's provider is the same entity as the payee's provider.
Open-Loop		A payment system or scheme designed for multiple providers to participate in. Payment system rules or national law may restrict participation to certain classes of providers.
Operations Risk Management		Tools to manage providers' risks in operating a DFS system.
Organization		Non-business An entity such as a business, charity or government department that uses mobile money as a service, e.g. taking bill payments, making bill payments and disbursing salaries
Over The Counter Services	OTC, Mobile to Cash	Services provided by agents when one end party does not have an eMoney account: the (remote) payer may pay the eMoney to the agent's account, who then pays cash to the non-account holding payee.
Participant		A provider who is a member of a payment scheme, and subject to that scheme's rules.
Partner Bank		Financial institution supporting the FSP and giving it access to the local banking ecosystem.
Payee	Receiver	The recipient of funds in a payment transaction.
Payee FSP		Payee's financial service providers.
Payer	Sender	The payer of funds in a payment transaction.
Payer FSP		Payer's financial service providers.
Paying for Purchases	C2B - Consumer to Business	Making payments from a consumer to a business: the business is the "payment acceptor" or merchant.
Payment System	Payment Network, Money Transfer System	Encompasses all payment-related activities, processes, mechanisms, infrastructure, institutions and users in a country or a broader region (eg a common economic area).
Payment System Operator	Mobile Money Operator, Payment Service Provider	The entity that operates a payment system or scheme.
Peer FSP		The counterparty Mobile Money Provider's Platform financial service

Term	Alternative and Related Terms	Mojaloop Definition
Mobile Money Platform		provider.
PEP		Politically Exposed Person. Someone who has been entrusted with a prominent public function. A PEP generally presents a higher risk for potential involvement in bribery and corruption by virtue of their position and the influence that they may hold (e.g. 'senior foreign political figure', 'senior political figure', 'foreign official', etc.).
Phone Number		Non identifying number associated with one or more end users as contact information for the end user. These numbers use the E.164 standard. Phone numbers are not required as a user number, though they can be used that way if a government or DFSP insists.
Platform	Payment Platform, Payment Platform Provider	A term used to describe the software or service used by a provider, a scheme, or a switch to manage end user accounts and to send and receive payment transactions.
Point of Sale Device	Terminal, Acceptance Device, POS, mPOS	Any device meant specifically for managing the receipt of electronic payments.
Posting	Clearing	The act of the provider of entering a debit or credit entry into the end user's account record.
Prefunding		The process of adding funds to Vostro/Nostro accounts.
Prepaid Cards		eMoney product for general purpose use where the record of funds is stored on the payment card (on magnetic stripe or the embedded integrated circuit chip) or a central computer system, and which can be drawn down through specific payment instructions to be issued from the bearer's payment card.
Processor	Gateway	An enterprise that manages, on an out-sourced basis, various functions for a digital financial services provider. These functions may include transaction management, customer database management, and risk management. Processors may also do functions on behalf of payments systems, schemes, or switches.
Promotion		FSP marketing initiative offering the user a transaction/service fee discount on goods or services. May be implemented through the use of a coupon.
Provider	Financial Service	The entity that provides a digital financial service to an end user (either a consumer, a business, or a government.) In a closed-loop payment system,

Term	Alternative and Related Terms	Mojaloop Definition
	Provider, Payment Service Provider, Digital Financial Services Provider	the Payment System Operator is also the provider. In an open-loop payment system, the providers are the banks or non-banks which participate in that system.
Pull Payments		A payment type which is initiated by the payee: typically a merchant or payment acceptor, whose provider "pulls" the funds out of the payer's account at the payer's provider.
Push Payments		A payment type which is initiated by the payer, who instructs their provider to debit their account and "push" the funds to the receiving payee at the payee's provider.
Quoting		The process a DFSP uses to ask for the fees and ILP packet from the destination
Reconciliation		Cross FSP Reconciliation is the process of ensuring that two sets of records, usually the balances of two accounts, are in agreement between FSPs. Reconciliation is used to ensure that the money leaving an account matches the actual money transferred. This is done by making sure the balances match at the end of a particular accounting period.
Recourse		Rights given to an end user by law, private operating rules, or specific agreements by providers, allowing end users the ability to do certain things (sometimes revoking a transaction) in certain circumstances.
Refund		A repayment of a sum of money.
Registration	Enrollment, Agent Registration	The process of opening a provider account. Separate processes are used for consumers, merchants agents, etc.
Regulator		A governmental organization given power through national law to set and enforce standards and practices. Central Banks, Finance and Treasury Departments, Telecommunications Regulators, and Consumer Protection Authorities are all regulators involved in digital financial services.
Reservation		Part of a 2-phase transfer operation in which the funds to be transferred are 'segregated' (i.e. made unusable) for a predetermined duration, commonly governed by a timeout period, to any other transfer attempts.
Reversal		The process of reversing a completed transfer.
Risk	Fraud	The practices that enterprises do to understand, detect, prevent, and manage

Term	Alternative and Related Terms	Mojaloop Definition
Management	Management	various types of risks. Risk management occurs at providers, at payments systems and schemes, at processors, and at many merchants or payments acceptors.
Risk-based Approach		A regulatory and/or business management approach that creates different levels of obligation based on the risk of the underlying transaction or customer.
Rollback		The process of reversing a completed transfer.
RTGS		Real time gross settlement
Rules		The private operating rules of a payments scheme, which bind the direct participants (either providers, in an open-loop system, or end users, in a closed-loop system).
Saving and Investing		Keeping funds for future needs and financial return
Savings Products		An account at either a bank or non-bank provider, which stores funds with the design of helping end users save money.
Scheme		A set of rules, practices and standards necessary for the functioning of payment services.
Secure Element		A secure chip on a phone that can be used to store payment data.
Security Level		Security specification of the system which defines effectiveness of risk protection.
Sending or Receiving Funds		Making and receiving payments to another person
Settlement		An act that discharges obligations in respect of funds or securities transfers between two or more parties.
Settlement System	Net Settlement, Gross Settlement, RTGS	A system used to facilitate the settlement of transfers of funds, assets or financial instruments. Net settlement system: a funds or securities transfer system which settles net settlement positions during one or more discrete periods, usually at pre-specified times in the course of the business day. Gross settlement system: a transfer system in which transfer orders are settled one by one.
Short Message Service		A service for sending short messages between mobile phones.

Term	Alternative and Related Terms	Mojaloop Definition
SIM Card	SIM ToolKit, Thin SIM	A smart card inside a cellular phone, carrying an identification number unique to the owner, storing personal data, and preventing operation if removed. A SIM Tool Kit is a standard of the GSM system which enables various value-added services. A "Thin SIM" is an additional SIM card put in a mobile phone.
Smart Phone		A device that combines a mobile phone with a computer.
Standards Body	EMV, ISO, ITU, ANSI, GSMA	An organization that creates standards used by providers, payments schemes, and payments systems.
Storing Funds	Account, Wallet	Keeping funds in secure electronic format. May be a bank account or an eMoney account.
Super Agent	Master agent	In some countries, agents are managed by Super Agents or Master Agents who are responsible for the actions of their agents to the provider.
Supplier Payment	B2B - Business to Business, B2G - Business to Government	Making a payment from one business to another for supplies, etc: may be in-person or remote, domestic or cross border. Includes cross-border trade.
SVA (Stored Value Account)		Accounts in which funds are kept in a secure, electronic format.
Switch		An entity which receives transactions from one provider and routes those transactions on to another provider. A switch may be owned or hired by a scheme, or be hired by individual providers. A switch will connect to a settlement system for inter-participant settlement.
Systemic Risk		In payments systems, the risk of collapse of an entire financial system or entire market, as opposed to risk associated with any one individual provider or end user.
Tax Payment	C2G, B2G	Making a payment from a consumer to a government, for taxes, fees, etc.
Tokenization		The use of substitute a token ("dummy numbers") in lieu of "real" numbers, to protect against the theft and misuse of the "real" numbers. Requires a capability to map the token to the "real" number.
Trading	International Trade	The exchange of capital, goods, and services across international borders or territories
Transaction Accounts		Transaction account is broadly defined as an account held with a bank or other authorized and/or regulated service provider (including a non-bank) which can be used to make and receive payments. Transaction accounts can be further differentiated into deposit transaction accounts and eMoney

Term	Alternative and Related Terms	Mojaloop Definition
		accounts. Deposit transaction account is a deposit account held with banks and other authorised deposit-taking financial institutions that can be used for making and receiving payments. Such accounts are known in some countries as current accounts, cheque accounts or other similar terms.
Transaction Cost		The cost to a DFS provider of delivering a digital financial service. This could be for a bundle of services (e.g. a "wallet") or for individual transactions.
Transfer		A general term for sending money. Local transfer refers to sending money within a ledger or DFSP, interledger transfers go between ledgers or DFSPs.
Trusted Execution Environment		An development execution environment that has security capabilities and meets certain security-related requirements.
Ubiquity		The ability of a payer to reach any (or most) payees in their country, regardless of the provider affiliation of the receiving payee. Requires some type of interoperability.
Unbanked	Underbanked, Underserved	Unbanked people do not have a transaction account. Underbanked people may have a transaction account but do not actively use it. Underserved is a broad term referring to people who are the targets of financial inclusion initiatives. It is also sometimes used to refer to a person who has a transaction account but does not have additional DFS services.
User Creation		A process for creating an individual user in the system.
User Number	User ID	A number that identifies an end user. We assume it uses <a href="#">E.164 format</a> . Depending on the country and DFSP, this can be a phone number, some form of DFSP provided ID or some form of national ID. The DFSPs associates the number to a phone number and a primary account number. Money is generally sent to the user number, and not directly to the account.
User On-boarding		A process for creating an individual user in the system and all additional related actions such as creation of user PIN, creation of user account, KYC data capture (photo, fingerprints), etc.
USSD		A communication technology that is used to send text between a mobile phone and an application program in the network.
Voucher		A token that entitles the holder to a discount or that may be exchanged for goods or services.
Wallet		Repository of funds for an account.
Whitelist		A list or register of entities (registered users) that are being provided a

Term	Alternative and Related Terms	Mojaloop Definition
		<p>particular privilege, service, mobility, access or recognition, especially those that were initially blacklisted. Entities on the list will be accepted, approved and/or recognized. Whitelisting is the reverse of blacklisting, the practice of identifying entities that are denied, unrecognized, or ostracized. Where entities are registered users (or user accounts, if granularity allows) and services are informational (e.g. balance check), transactional (e.g. debit/credit) payments services or lifecycle (e.g. registration, closure) services.</p>
'x'-initiated		<p>Used when referring to the side that initiated a transaction, e.g. agent-initiated cash-out vs. user-initiated cash-out"</p>

# Evolution of Mojaloop

Here we document the reasoning behind certain tools, technology and process choices for Mojaloop.

- **Open source** - the entire project may be made open source in accordance with the [level one principles](#). All tools and processes must be open source friendly and support an Apache 2.0 license and no restrictive licenses.
- **Agile development** - The requirements need to be refined as the project is developed, therefore we picked agile development over waterfall or lean.
- **Scaled Agile Framework** - there are four initial development teams that are geographically separate. To keep the initial phase of the project on track, the [scaled agile framework \(SAFe\)](#) was picked. This means work is divided into program increments (PI) that are typically four 2 week sprints long. As with the sprints, the PI has demo-able objective goals defined in each PI meeting.
- **Threat Modeling, Resilience Modeling, and Health Modeling** - because this code needs to exchange money in an environment with very flaky infrastructure it must have good security, resilience, and easily report its health state and automatically attempt to return to it. To achieve this we employ basic tried and true [modeling practices](#).
- **Automated Testing** - for the most part, most testing will be automated to allow for easy regression. See the [automated testing strategy](#).
- **Microservices** - Because the architecture needs to easily deploy, scale, and have components be easily replaced or upgraded it will be built as a set of microservices.
- **APIs** - In order to avoid confusion from too many changing microservices, we use strongly defined APIs. APIs will be defined using [OpenAPI](#) or [RAML](#). Teams document their APIs with Swagger v2.0 or RAML v0.8 so they can automatically test, document, and share their work. Swagger is slightly preferred as there are free tools. Mule will make use of RAML 0.8. Swagger can be automatically converted to RAML v0.8, or manually to RAML v1.0 if additional readability is desired.
- **Services** - Microservices are grouped and deployed in a few services such as the DFSP, Central Directory, etc. Each of these will have simple defined interfaces, configuration scripts, tests, and documentation.
- **Database Storage** - although Microsoft SQL Server is widely used in Africa, we need a SQL backend that is open source friendly and can scale in a production environment. Thus, we chose PostgreSQL. The database is called through an adapter and the stored procedures are kept in simple ANSI SQL so that it can be replaced later with little trouble.
- **USSD** - Smart phones are only 25% of the target market and not currently supported by most money transfer service, so we need a protocol that will work on simple feature phones. Like M-Pesa, we are using USSD between the phone and the digital financial service provider (DFSP).

- **Operating System** - Again, Microsoft Windows is widely used in many target countries, but we need an operating system that is free of license fees and is open source compatible. We are using Linux. We don't have a dependency on the particular flavor, but are using the basic Amazon Linux. In the Docker containers, Alpine Linux is used.
- **Interledger** - The project needed a lightweight, open, and secure transport protocol for funds. Interledger.org provides all that. It also provides the ability to connect to other systems. We also considered block chain systems, but block chain systems send very large messages which will be harder to guarantee delivery of in third world infrastructure. Also, while blockchain systems provide good anonymity, that is not a project goal. To enable fraud detection, regulatory authorities need to be able to request records of transfers by account and person.
- **MuleSoft** - For the most part, the mule server is simple a host and pass through for Level One Client API calls. However, it will be necessary to deploy Mojaloop system into existing financial providers. MuleSoft provides an excellent adapter so that the APIs can be easily hooked up to existing systems while providing cross cutting concerns like logging, fault tolerance, and security. The core pieces used don't require license fees.
- **NodeJS** - NodeJS is designed to create simple microservices and it has a huge set of open source libraries available. Node performance is fine and while Node components don't scale vertically a great deal, we plan to scale horizontally, which it does fine. The original Interledger code was written in NodeJS as was the level one prototype this code is based on. Most teams used Node already, so this made sense as a language.
- **NodeJS "Standard"** - Within NodeJS code, we use Standard as a code style guide and to enforce code style.
- **Java** - Mule can't run NodeJS directly, so some adapters to mule and interop pieces are written in Java. This is a very small part of the overall code.
- **Checkstyle** - Within Java code, we use Checkstyle as a code style guide and style enforcement tool.
- **GitHub** - GitHub is the standard source control for open source projects so this decision was straightforward. We create a story every time for integration work. Create bugs for any issues. Ensure all stories are tracked throughout the pipeline to ensure reliable awx.
- **Slack** - Slack is used for internal team communication. This was largely picked because several team already used it and liked it as a lightweight approach compared to email.
- **ZenHub** - We needed a project management solution that was very light weight and cloud based to support distributed teams. It had to support epics, stories, and bugs and a basic project board. VS and Jira online offerings were both considered. For a small distributed development team an online service was better. For an open source project, we didn't want ongoing maintenance costs of a server. Direct and strong GitHub integration was important. It was very useful to track work for each microservice with that microservice. Jira and VS both have more overhead than necessary for a project this size and don't integrate as cleanly with GitHub as we'd want. ZenHub allowed us to start work immediately. A disadvantage is the lack of support for cumulative flow diagrams and support for tracking # of stories instead of points, so we do these manually with a spreadsheet updated daily and the results published to the "Project Management" Slack channel (Cumulative flow is being added to Zenhub, but wasn't available for most of the project).

- **AWS** - We needed a simple hosting service for our Linux instances, and we aren't going to use many of the extra services like geo-redundancy, since we expect customers may wish to self-host. AWS is an industry standard and works well. We considered Azure, which also would have worked, but it's harder for the Gates Foundation to get an Azure subscription than AWS.
- **Docker** - the project needs to support both local and cloud execution. We have many small microservices that have very simple specific configurations and requirements. The easiest way to guarantee that the service works the same way in every environment from local development, to cloud, to hosted production is to put each microservice in a Docker container along with all the prerequisites it needs to run. The container becomes a secure, closed, pre-configured, runnable unit.
- **CircleCI** - to get started quickly we needed an online continuous build and testing system that can work with many small projects and a distributed team. Jenkins was considered, but it requires hosting a server and a lot of configuration. CircleCI allowed for a no host solution that could be started with no cost and very limited configuration. We thought we might start with CircleCI and move off later if we outgrew it, but that hasn't been needed.
- **Artifactory** - After the build we need private repository to put our NodeJS packages and Docker containers until they are formally published. Docker and AWS both do this, and any solution would work. We chose Artifactory from JFrog simply because one team already had an account with it and had it setup.
- **SonarQube** - We need an online dashboard of code quality (size, complexity, issues, and coverage) that can aggregate the code from all the repos. We looked at several online services (Codecov, Coveralls, and Code Climate), but most couldn't do complexity or even number of lines of code. Code Climate has limited complexity (through ESLint), but costs 6.67/seat/month. SonarQube is free, though it required us to setup and maintain our own server. It gave the P1 features we wanted.
- **Markdown** - Documentation is a deliverable for this project, just like the code, and so we want to treat it like the code in terms of versioning, review, check in, and tracking changes. We also want the documentation to be easily viewable online without constantly opening a viewer. GitHub has a built-in format called Markdown which solves this well. The same files work for the Wiki and the documents. They can be reviewed with the check in using the same tools and viewed directly in GitHub. We considered Google Docs, Word and PDF, but these binary formats aren't easily diff-able. A disadvantage is that markdown only allows simple formatting - no complex tables or font changes - but this should be fine when our main purpose is clarity.
- **Draw.io** - We need to create pictures for our documents and architecture diagrams using an (ideally free) open source friendly tool, that is platform agnostic, supports vector and raster formats, allows WYSIWYG drawing, works with markdown, and is easy to use. We looked at many tools including: Visio, Mermaid, PlantUML, Sketchboard.io, LucidChart, Cacoo, Archi, and Google Drawings. Draw.io scored at the top for our needs. It's free, maintained, easy to use, produces our formats, integrates with DropBox and GitHub, and platform agnostic. In order to save our diagrams, we have to save two copies - one in SVG (scalable vector) format and the other in PNG (raster). We use the PNG format within the docs since it can be viewed directly in GitHub. The SVG is used as the master copy as it is editable.
- **Dactyl** - We need to be able to print the online documentation. While it's possible to print

markdown files directly one at a time, we'd like to put the files into set of final PDF documents, where one page might end up in more than one final manual. [Dactyl](#) is a maintained open source conversion tool that converts between markdown and PDF. We originally tried Pandoc, but it had bugs with converting tables. Dactyl fixes that and is much more flexible.

- **Ansible** - We need a way to set microservice configurations and monitor/verify that those configuration are correct. We need the tool to be very simple to setup and use. It must support many OS's and work for both the cloud and local environments as well as Docker and non-Docker setup. We looked at many tools including: Chef, Puppet, Cloud Formation, Docker Compose/Swarm, Kubernetes, Terraform, Salt, and Ansible. Chef and Puppet were eliminated because they have a very large learning curve and large setup requirements. AWS Cloud Formation and Docker were both limited to specific environments and we need broader support. Terraform was a good tool, but works differently with each environment, so a configuration for cloud can't be reused locally. Kubernetes is also good, but is designed to send commands across large scale environments, not configure a few specific microservices. Salt and Ansible can both do the job. Salt is more scalable and performant, as it puts an agent on each server to orchestrate config. Ansible is much simpler, having an agentless direct setup. We went with Ansible because of the simple setup and learning curve. We don't need the speed and scale Salt provides and accept a slightly lower performance. Ansible allows us to define the expected state of the microservice in a playbook. If the state is incorrect, Ansible can automatically alert and correct the state. In this way it is both a monitoring and configuration tool.

# Exporting the Documentation

In Mojaloop, the source for the documentation are markdown files stored in GitHub. We use a tool called [Dactyl](#) to convert files from markdown (md) format to PDF format. The PDF format is the exported format we use to share offline documentation. Overview and cross-repo documentation is in the Docs repository. Other repositories have detailed information about their contents.

## Setup

See [Dactyl setup](#) to setup the tool. Dactyl has dependencies on Python and on a command line tool [Prince](#) to do part of that conversion.

All the repositories should be in the same root directory.

Because building the documentation requires md files from multiple repos, you need to pull the latest files from all the repos mentioned in the dactyl-config file. The list of required repos may change, but currently includes all of the following:

- [mojaloop](#)
- [central-direcotry](#)
- [central-fraud-sharing](#)
- [central-ledger](#)
- [docs \(this repository\)](#)
- [ilp-service](#)
- [forensic-logging-sidecar](#)
- [interop-ilp-ledger](#)
- [interop-dfsp-direcotry](#)

You can use the following script to clone all of the needed repositories:

```
For repo in mojaloop central-direcotry central-fraud-sharing central-ledger docs ilp-service forensic-logging-sidecar interop-ilp-ledger interop-dfsp-direcotry; do git clone git@github.com:mojaloop/"$repo".git; done
```

Later, you can update all the repositories to the latest with the following:

```
for repo in mojaloop central-directory central-fraud-sharing central-ledger docs ilp-service forensic-logging-sidecar interop-ilp-ledger interop-dfsp-directory; do cd "$repo"; git pull; cd ..; done
```

# Build Process

## Copy the image files

Images that are going to be included in the documentation need to be copied to a images directory. Run the script CopyImages.sh from the root directory (the one above docs)

```
$./docs/ExportDocs/CopyImages.sh
```

## Run Dactyl

From the root directory run the dactyl build command for the document. For example, to generate a full set of documents run:

```
$ dactyl_build -t all -c docs/ExportDocs/dactyl-config.yml --pdf
```

To generate just the stakeholder overview run:

```
$ dactyl_build -t stakeholder -c docs/ExportDocs/dactyl-config.yml --pdf
```

# Build Info

Dactyl first converts all the md files to HTML. In that process it can apply common css styles and cover pages which are in the pdf\_templates directory. Dactyl uses [Prince](#) to convert the HTML files to PDF. The --leave\_temp\_files parameter can be very useful for debugging if you want to see the intermediate HTML.

The files are written to the out/ directory under the root.