# Mowali Guide to Golden DFSP Simulator Version 1

Version 0.3

Publication date: 26 Jun, 2019

# Table of Contents

# Summary of changes for "Mowali Guide to Golden DFSP Simulator Version 1"

## Changes between the versions published on 20 June and 26 June

- Configuring the simulator backend: Updated default value of `OUTBOUND_ENDPOINT`.
- Configuring the SDK (Scheme Adapter): Updated default value of `PEER_ENDPOINT`. Added `INBOUND_PORT` and `OUTBOUND_PORT`.
- Creating parties for testing purposes: Updated instructions on how to find out the IP address of the simulator.
- Testing as a Payer DFSP: Updated instructions on how to find out the IP address of the simulator.
- Testing as a Payer DFSP: Updated instructions on how to find out the IP address of the simulator.

## Changes between the versions published on 18 June and 20 June

- Introduction to Mowali Golden DFSP Simulator v1: Added "Mowali Rules" to Figure 1 and clarified information around configuration files.
- Getting started with Golden DFSP Simulator v1: Added entry for `install.sh` file.
- Configuring the simulator backend: Added default values.
- Configuring the SDK (Scheme Adapter): Added default values.
- Setting up certificates and security: Added information about default set of keys.
- Testing as a Payer DFSP: Updated container names to look for in the output of the `docker network inspect` command.
- Testing as a Payer DFSP: Updated container name to look for in the output of the `docker network inspect` command.

## Changes between the versions published on 06 June and 18 June

- Getting started with Golden DFSP Simulator v1: Updated names of `sim-backend.env` and `scheme-adapter.env` files.
- Configuring Golden DFSP Simulator v1: Updated names of `sim-backend.env` and `scheme-adapter.env` files.
- Configuring the simulator backend:
  - Updated name of `sim-backend.env` file.
  - Added entry for `RULES_FILE`.
- Configuring the SDK (Scheme Adapter): Updated name of `scheme-adapter.env` file.
- Starting up Golden DFSP Simulator v1: Updated steps to reflect presence of `install.sh` script.
- Creating parties for testing purposes: Updated project name in `docker network inspect` command.
- Testing as a Payer DFSP: Updated project name in `docker network inspect` command.
- Testing as a Payer DFSP: Updated project name in `docker network inspect` command.

# Introduction to Mowali Golden DFSP Simulator v1

Mowali provides simulators that allow DFSPs to interact with an emulated Mowali Hub and peer DFSP implementation. The aim of Mowali simulators is to ensure a swift and smooth onboarding process, as well as to facilitate the qualification of DFSPs before allowing them into the production environment.

Simulators come in various versions and DFSPs interact with a version of the simulator at various stages of the onboarding journey. Golden DFSP Simulator version 1 is provided to DFSPs by Mowali as a downloadable file, allowing them to run tests locally.

The Mowali Golden DFSP Simulator v1 supports making happy path `/participants`, `/parties`, `/quotes`, `/transfers` , and `/reports` calls to an emulated Mowali Hub and peer DFSP. In addition to happy path scenarios, error cases are also supported around header and JSON schema validation. Validation with regards to specific rules (for example, sending more money than the permitted amount) is also supported.

Golden DFSP Simulator v1 is provided as a single Docker image, which itself is made up of three Docker containers:

- Mojaloop SDK, acting as the emulated Mowali Hub:
    - Standard Components to implement:
        - Mojaloop-specification-compliant security:
            - two-way TLS with mutual X.509 authentication
            - JSON Web Signature (JWS) signing of messages
            - generation of the Interledger Protocol (ILP) packet with signing and validation
        - HTTP headers and header processing
    - Scheme Adapter to allow presenting a simplified version of the Mojaloop API to the simulator backend
- Simulator Docker Container: The simulator backend, including the simulation of a peer DFSP. The API between the simulated peer DFSP and the SDK is a simplified version of the Mojaloop API. A Swagger specification of the simplified API can be found in GitHub.
- Cache Docker Container: A fast in-memory database.

The three containers are run together, they are configured to work as a single simulator unit.

The following figure provides an illustration of the scenario when a DFSP is testing against Golden DFSP Simulator v1.

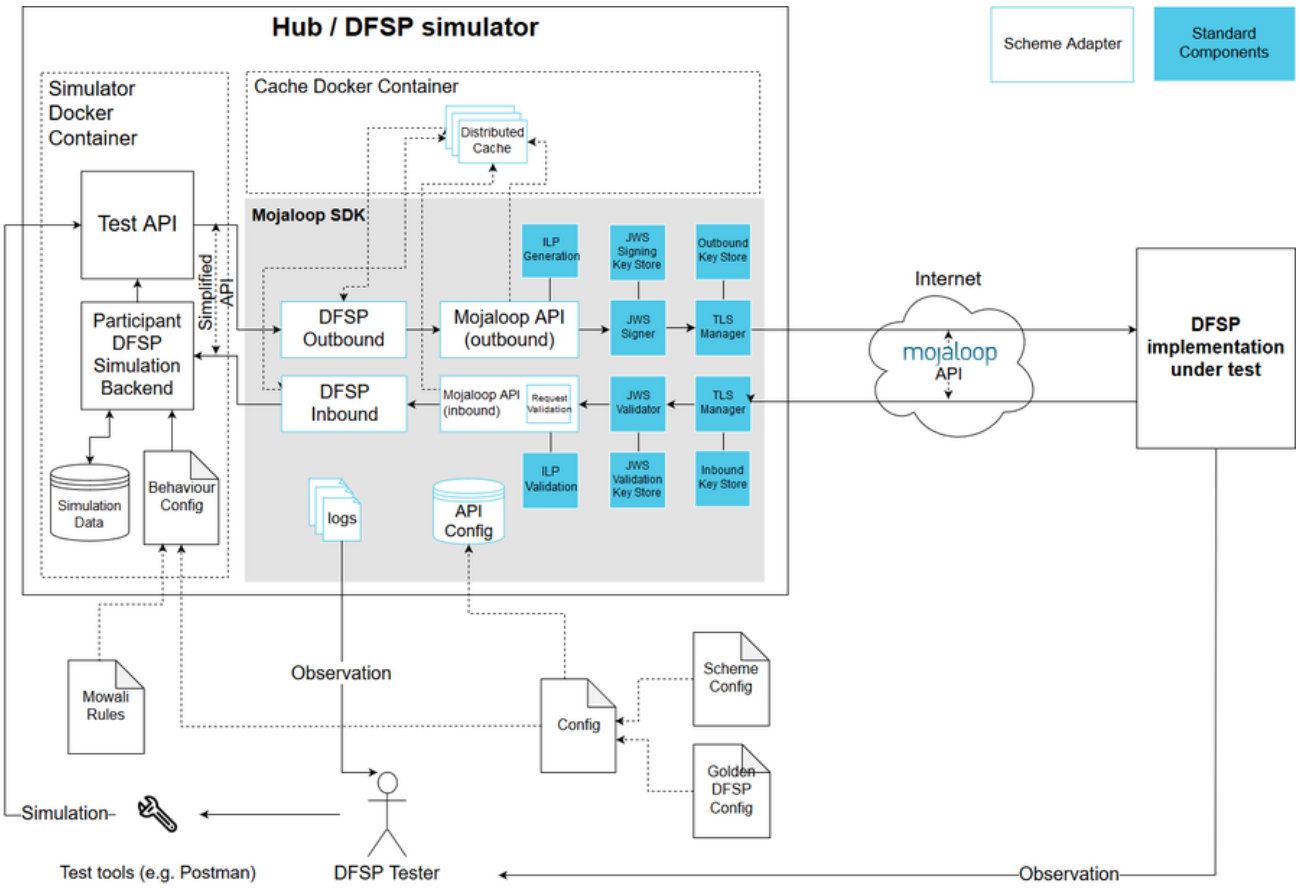


**Figure 1: DFSP tests against simulator v1**

Some of the key components that DFSPs can configure and interact with are:

- **Scheme Config**: A configuration file (`scheme-adapter.env`) to program the simulator with, it controls the behaviour of the Mojaloop SDK container. Supplied by Mowali.
- **Golden DFSP Config**: A configuration file (`sim-backend.env`) to program the simulator with, it controls the behaviour of the Simulator Docker Container. Supplied by Mowali.
- **Config** – a combination of Scheme Config and Golden DFSP Config: A configuration file that is used to control the behaviour of the simulator and certain technical configuration options. The configuration provided in this file ends up being set in the **Behaviour Config** and **API Config** components in the containers.
- **Mowali Rules**: A configuration file (`mowali_rules.json`) to program the simulator with, it controls the behaviour of the Simulator Docker Container. Supplied by Mowali. The configuration provided in this file ends up being set in the **Behaviour Config** component in the Simulator Docker Container. An example of the behaviour that Mowali Rules can control is the type of money transfers that are allowed. For example, if a merchant payment is attempted and the configuration only allows P2P transfers, the DFSP will get an error.
- **logs**: The Docker output logs of the SDK. They can be observed by the DFSP Tester, for example, for errors.
- **Simulation Data**: Simulator backend logs stored in a database on the file system.
- **Test API**: Allows for manipulating the simulated peer DFSP so that a DFSP can test being both on the Payer and Payee end of a transfer flow. If a DFSP wants to test for receiving money, they will need to trigger an incoming transfer via the Test API. This will prompt the Test API to construct a transfer message, which will get translated through the SDK into different API messages that have to be exchanged across the Mojaloop interface in order to make a money transfer.

# Getting started with Golden DFSP Simulator v1

The Golden DFSP Simulator v1 package is provided to DFSPs by Mowali. It contains the following:

| Item | Details |
|---|---|
| Docker image | The Docker image of the simulator application. |
| `docker-compose.yml` file | The simulator is a multi-container application, and as such, it requires Docker Compose to define and run the containers that make up the application.<br><br>With Compose, you use a YAML file, `docker-compose.yml`, to define your application's services so they can be run together. `docker-compose.yml` references all three containers that make up the simulator, including the environment files (`scheme-adapter.env` and `sim-backend.env`) that load the configuration into the containers when starting up the simulator application.<br><br>For details on how to start up the simulator, see Starting up Golden DFSP Simulator v1. |
| `sim-backend.env` file | This is the Golden DFSP Config file, which controls the behaviour of the Simulator Docker Container (that is, the simulator backend and the simulated peer DFSP).<br><br>For details on how to configure the simulator using the `sim-backend.env` file, see Configuring the simulator backend. |
| `scheme-adapter.env` file | This is the Scheme Config file, which controls the behaviour of the Scheme Adapter in the Mojaloop SDK container.<br><br>For details on how to configure the simulator using the `scheme-adapter.env` file, see Configuring the SDK (Scheme Adapter). |
| `install.sh` | A script that imports the Docker image. |

**Table 1: Contents of Golden DFSP Simulator v1 package**

To get started using the simulator, follow these steps:

| Step | Details |
|---|---|
| 1 | Download and save the image locally. |
| 2 | Configure the simulator. |
| 3 | Write transfer logic rules. |
| 4 | Set up certificates and security. |
| 5 | Start up the simulator. |
| 6 | Create parties for testing purposes. |
| 7 | Test your DFSP implementation by making calls to the simulator. |

**Table 2: Getting started with Golden DFSP Simulator v1**

# Configuring Golden DFSP Simulator v1

Configuring the simulator is done through Docker environment files. Environment files store configuration options, which are loaded into the various containers when starting up the simulator.

The following environment files are provided:

- `sim-backend.env` file: This is the Golden DFSP Config file, which controls the behaviour of the Simulator Docker Container (that is, the simulator backend and the simulated peer DFSP). For details, see Configuring the simulator backend.
- `scheme-adapter.env` file: This is the Scheme Config file, which controls the behaviour of the Mojaloop SDK container (that is, the simulated Hub). For details, see Configuring the SDK (Scheme Adapter).

Both files come with default values.

## Configuring the simulator backend

The `sim-backend.env` file allows you to configure the following environment variables, which control the behaviour of the Simulator Docker Container:

| Option | Type | Default value | Description |
|---|---|---|---|
| `MUTUAL_TLS_ENABLED` | boolean | `false` | Enable mutual TLS authentication. Useful when the simulator is not running in a Mowali environment, that is, when running it locally against your own implementation. |
| `HTTPS_ENABLED` | boolean | `false` | Enable server-only TLS; that is, serve on HTTPS instead of HTTP. |
| `CA_CERT_PATH` | string | `./secrets/cacert.pem` | Location of the Certificate Authority's (CA) root certificate required for TLS. |
| `SERVER_CERT_PATH` | string | `./secrets/servercert.pem` | Location of the TLS server public key. |
| `SERVER_KEY_PATH` | string | `./secrets/serverkey.pem` | Location of the TLS server private key. |
| `LOG_INDENT` | integer | `0` | The number of space characters by which to indent pretty-printed logs. If set to `0`, log events will each be printed on a single line. |
| `SQLITE_LOG_FILE` | string | `:memory:` | The name of the Simulation Data SQLite log file.<br><br>Setting `SQLITE_LOG_FILE=:memory:` will use an in-memory SQLite database, which will be faster and not consume disk space. However, it will also mean that the logs will be lost once the container is stopped.<br><br>To save the logfile, change `:memory:` to a file path. |
| `SCHEME_NAME` | string | `golden` | The `fspId` of the simulated peer DFSP. Note that this is the same as `DFSP_ID`.<br><br>The simulator will accept any requests routed to `FSPIOP-Destination: $SCHEME_NAME`. Other requests will be rejected. |

| | | | |
|---|---|---|---|
| `MODEL_DATABASE` | string | `./model.sqlite` | The name of the SQLite model database, which contains data related to the API calls made via the simulator. If you would like to start the simulator with a pre-loaded state, you can use a pre-existing file. If running in a container, you can mount an SQLite file as a volume in the container to preserve state between runs.<br><br>Use `MODEL_DATABASE=:memory:` for an ephemeral in-memory database. When using this option, all data will be lost once the container is stopped. |
| `OUTBOUND_ENDPOINT` | string | `http://scheme-adapter:4001` | The endpoint that the Test API uses to make outbound `/transfers` requests. (It might be a container in the compose file so remember the networking IP.)<br><br>If you want to trigger a request into your DFSP backend from the simulator, the Simulator Docker Container will call the Mojaloop SDK API at this endpoint. |
| `DFSP_ID` | string | `golden` | The `fspId` of the simulated peer DFSP. Note that this is the same as `SCHEME_NAME`.<br><br>The simulator will accept any requests routed to `FSPIOP-Destination: $DFSP_ID`. Other requests will be rejected. |
| `FEE_MULTIPLIER` | float | `0.05` | When the simulator gets a request for a quote from an inbound call, it can automatically add some fees to test the DFSP's fee implementation in their quotes handling.<br><br>Set it to a number between `0` and `1`, and the transfer amount will be multiplied by this number. The outcome will be applied as the fee. For example, in the case of a transfer of 100 USD, a `FEE_MULTIPLIER` of `0.1` reuslts in fees of USD 10 being applied to the quote response.<br><br>**NOTE:** Currently this option is not in use as fees are not supported. Set it to `0`. |
| `RULES_FILE` | string | `./mowali-rules.json` | Specifies the location of a rules file for the simulator backend. Rules can be used to produce specific simulator behaviours in response to incoming requests that match certain conditions – for example, a rule can be used to trigger NDC errors for transfers between certain limits. |

**Table 3: Simulator backend (Golden DFSP Config) configuration options**

## Configuring the SDK (Scheme Adapter)

The `scheme-adapter.env` file allows you to configure the following environment variables, which control the behaviour of the Mojaloop SDK container:

| Option | Type | Default value | Description |
|---|---|---|---|
| `INBOUND_PORT` | integer | `4000` | Port number that the inbound (Mojaloop API) HTTP server will listen on. |
| `OUTBOUND_PORT` | integer | `4001` | Port number that the outbound (simplified DFSP outbound API) HTTP server will listen on. |

| | | | |
|---|---|---|---|
| `MUTUAL_TLS_ENABLED` | boolean | `false` | Enable mutual TLS authentication. Useful when the simulator is not running in a secure environment, that is, when running it locally against your own implementation. |
| `VALIDATE_INBOUND_JWS` | boolean | `false` | Enable JSON Web Signature (JWS) verification. |
| `JWS_SIGN` | boolean | `true` | Enable JWS signing. |
| `JWS_SIGNING_KEY_PATH` | string | `/jwsSigningKey.key` | Path to JWS private key of the simulated peer DFSP used for signing outbound messages. |
| `JWS_VERIFICATION_KEYS_DIRECTORY` | string | `/jwsVerificationKeys` | Path to JWS public key of your DFSP implementation.<br><br>A verification key is needed for every DFSP that might send you a request. You get one by default, but if you have your own signing key in your DFSP backend that you want to test, load your public key into the simulator so you can test if your JWS signing is working as expected. |
| `CA_CERT_PATH` | string | `./secrets/cacert.pem` | Location of the Certificate Authority's (CA) root certificate required for TLS. |
| `SERVER_CERT_PATH` | string | `./secrets/servercert.pem` | Location of the TLS server public key. |
| `SERVER_KEY_PATH` | string | `./secrets/serverkey.pem` | Location of the TLS server private key. |
| `LOG_INDENT` | integer | `0` | The number of space characters by which to indent pretty-printed logs. If set to `0`, log events will each be printed on a single line. |
| `SCHEME_NAME` | string | `default` | The `fspId` of the simulated peer DFSP. Note that this is the same as `DFSP_ID`.<br><br>The simulator will accept any requests routed to `FSPIOP-Destination: $SCHEME_NAME`. Other requests will be rejected. |
| `CACHE_HOST` | string | `redis` | Host information for the Cache Docker Container. |
| `CACHE_PORT` | integer | `6379` | Port information for for the Cache Docker Container. |
| `PEER_ENDPOINT` | string | `172.21.0.5:4000` | Endpoint of your DFSP implementation under test (where callbacks will be sent by the simulator). |
| `BACKEND_ENDPOINT` | string | `sim:3000` | Endpoint of the simulated peer DFSP, that is, the Participant DFSP Simulation Backend. |

| | | | |
|---|---|---|---|
| `DFSP_ID` | string | `mojaloop-sdk` | The `fspId` of the simulated peer DFSP. Note that this is the same as `SCHEME_NAME`.<br><br>The simulator will accept any requests routed to `FSPIOP-Destination: $DFSP_ID`. Other requests will be rejected. |
| `ILP_SECRET` | string | `Quaixohyaesahju3thivuiChai5cahng` | Secret used for generation and verification of secure ILP. |
| `EXPIRY_SECONDS` | integer | `60` | Expiry period in seconds for quotes and transfers issued by the SDK. Corresponds to `"expiration"` in the Mowali API. |
| `AUTO_ACCEPT_QUOTES` | boolean | `false` | If set to `false`, the SDK will not automatically accept all returned quotes but will halt the transfer after a quote response is received. This gives the the initiator of the transaction the opportunity to examine the quote before proceeding to the transfer stage. A further confirmation call will be required to complete the final transfer stage. |
| `CHECK_ILP` | boolean | `true` | Set to `true` to validate ILP, otherwise `false` to ignore ILP. |

**Table 4: SDK (Scheme Config) configuration options**

# Writing transfer logic rules

The simulator Docker image comes with a `mowali_rules.json` file, which acts as a rules engine and enables you to define arbitrary rules that will accept or reject transfers. By default, the file contains a single rule, which you can replace or extend with your own rules to test how various scenarios are handled.

The default rule specifies the following scenario: if the transfer amount is between 100 and 200 currency units, then the simulator will reply with an "NDC exceeded" error.

```
[{
 "conditions": {
  "all": [{
   "fact": "path",
   "operator": "equal",
   "value": "/transfers"
  }, {
   "fact": "body",
   "operator": "numberStringGreaterThanInclusive",
   "value": 100,
   "path": ".amount"
  }, {
   "fact": "body",
   "operator": "numberStringLessThanInclusive",
   "value": 200,
   "path": ".amount"
  }]
 },
 "event": {
  "type": "ndc-exceeded",
  "params": {
   "statusCode": 500,
   "body": {
                "statusCode": "4001",
                "message": "Payer FSP insufficient liquidity"
            }
  }
 }
}]
```

As you can see in the example, a rule is defined as an array with a `conditions` object and an `event` object. A rule specifies that if certain conditions (in the example above, all of the conditions) are true, then the specified event will be generated.

Reading the example:

- IF the path of the API call is equal to `/transfers`
- AND the `.amount` property on the body is greater than or equal to 100
- AND the `.amount` property on the body is less than or equal to 200
- THEN generate an "NDC exceeded" error, sending a response specified by the parameters

Note that the error returned in the example is not a Mojaloop API error. It is an error response that is specified in the Swagger file of the Simplified API, this is the API that the SDK (the Scheme Adapter) exposes to the simulator backend. This Simplified API error will in turn generate the corresponding Mojaloop API error. Using this rule, you can test how your system handles the case when you have run out of liquidity, and check whether the right error message is being sent.

The rules engine used by the simulator is an off-the-shelf rules engine, called json-rules-engine. For detailed information on how to write rules, see the json-rules-engine documentation.

Some basic guidelines are provided in this document as well:

- `fact`: The available values that your rule can use to decide what to do. Note that in the simulator, this is always the path of the incoming request and the body of the incoming request. This means that you can write a rule based

on the type of request that is coming in and based on what the properties of that request are. For information on what properties are available, see the *API services* section in the *Mowali API Guide for DFSPs* and check the data elements that are sent in the API calls.

- `event`: The rules can generate more than one event if more than one rule matches. However, the simulator will only ever return the first match.
    - The `event` must always have a `type` and `params`, and the `params` must always have a response `body`.
    - The `params` in the response `body` must always match what the Simplified API Swagger file specifies as the expected response to the Scheme Adapter Simplified API call that you are responding to. In the example above, what you should send is an error response to a transfer request.

# Setting up certificates and security

The simulator gets deployed with a default set of keys in the package.

> **NOTE:** Information about generating and signing your own TLS and JWS certificates and plugging them into the simulator will be provided in a future version of the document.

# Starting up Golden DFSP Simulator v1

Golden DFSP Simulator v1 is provided as a Docker image. To run an instance of the image, complete the following steps.

**Prerequisites:**

- Ensure you have Docker installed in your local environment.
- Ensure you have Docker Composer (a tool for running multi-container Docker applications) installed in your local environment.
- Familiarity with Docker is advised.

**Steps:**

1. Download the image provided by Mowali.
2. Unzip the `.tar` file.
3. Run the `install.sh` script. This imports the image.
4. Start up the container from the directory where the image is.

```
docker-compose up
```

# Creating parties for testing purposes

Before executing tests, add test parties to your own DFSP implementation and the simulator backend.

To create a party in the simulator backend, make an HTTP `POST repositories/parties` request to the Test API.

First, find out where you can reach the Test API. When starting up the simulator, `docker-compose up` will generate some IP addresses on a local Docker network. To find out the IP address of the simulator:

1. Find out the name of the simulator Docker network:
   - Execute `docker network ls` and look for the network with `mowali simulator` in its name.

     OR
   - Execute `docker-compose up` and note down the name of the network being created.
2. Execute: `docker network inspect <Docker network name>`
3. Look for the IPv4 address of the entry called `mowalisimulator_sim_1`. The default port of the Test API is `3003`.

The following is an example of the call to make to the Test API:

```
curl -X POST http://<simulator IP address>:3003/repositories/parties 'Content-Type:
application/json' -d '{
    "displayName": "Henrik Karlsson",
    "firstName": "Henrik", "middleName":
    "Johannes", "lastName": "Karlsson",
    "dateOfBirth": "1966-06-16",
    "idType": "MSISDN",
    "idValue": "1234567890" }'
```

The elements of the JSON object are:

| Element | Type | Description |
|---|---|---|
| `displayName` | string of 1-128 characters | Display name of the Party.<br>The regular expression is:<br>`^(?!\s*$)[\w .,'-]{1,128}$` |
| `firstName` | string of 1-128 characters | Party's first name.<br>The regular expression is:<br>`^(?!\s*$)[\w .,'-]{1,128}$` |
| `middleName` | string of 1-128 characters | Party's middle name.<br>The regular expression is:<br>`^(?!\s*$)[\w .,'-]{1,128}$` |
| `lastName` | string of 1-128 characters | Party's last name.<br>The regular expression is:<br>`^(?!\s*$)[\w .,'-]{1,128}$` |

| | | |
|---|---|---|
| `dateOfBirth` | string | Party's date of birth in the format `YYYY-MM-DD`.<br><br>The regular expression is:<br><br>`^(?:[1-9]\d{3}-(?:(?:0[1-9]\|1[0-2])-(?:0[1-9]\|1\d\|2[0-8])\|(?:0[13-9]\|1[0-2])-(?:29\|30)\|(?:0[13578]\|1[02])-31)\|(?:[1-9]\d(?:0[48]\|[2468][048]\|[13579][26])\|(?:[2468][048]\|[13579][26])00)-02-29)$` |
| `idType` | enumeration | Type of the identifier. Possible values are:<br><br>• `MSISDN`: Mobile Station International Subscriber Directory Number; that is, a phone number of Party.<br>The MSISDN identifier should be in international format according to the ITU-T E.164 standard.<br>Optionally, the MSISDN may be prefixed by a single plus sign, indicating the international prefix.<br>• `ACCOUNT_ID`: A bank account number or DFSP account ID should be used as reference to a participant. The `ACCOUNT_ID` identifier can be in any format, as formats can greatly differ depending on country and DFSP. |
| `idValue` | string of 1-128 characters | Identifier of the party. |

**Table 5: POST repositories/parties – elements of JSON object**

The following table lists the HTTP response status codes that the API supports:

| Status code | Reason | Description |
|---|---|---|
| 204 | No Content | The party was created |
| 400 | Bad Request | The request was malformed |
| 500 | Internal Server Error | An error occurred processing the request |

**Table 6: HTTP response status codes**

# Making test API calls using Golden DFSP Simulator v1

Golden DFSP Simulator v1 supports making test API calls to the following endpoints:

- `/participants`
- `/parties`
- `/quotes`
- `/transfers`
- `/reports` to retrieve daily transaction reports (Report 311)

Both happy path scenarios and error cases are supported. Validation is done on:

- schema and headers
- the transfer logic rules you have specified in `mowali-rules.json`
- JWS inbound signatures
- TLS server certificates

For information on functional test cases that you can run against the simulator, see the *Mowali Test Suite for DFSPs* document.

It is important to make a distinction between testing as a DFSP on the Payer side as opposed to a DFSP on the Payee side. For details, see Testing as a Payer DFSP and Testing as a Payee DFSP.

## Testing as a Payer DFSP

Making test calls as a Payer DFSP is a straightforward process, you can send requests to the simulator as you would to the Hub.

First, find out the endpoint where your DFSP implementation should send the request to. When starting up the simulator, `docker-compose up` will generate some IP addresses on a local Docker network. To find out the IP address:

1. Find out the name of the simulator Docker network:
   - Execute `docker network ls` and look for the network with `mowali simulator` in its name.

     OR
   - Execute `docker-compose up` and note down the name of the network being created.
2. Execute: `docker network inspect <Docker network name>`
3. Look for the IPv4 address of the entry called:
   - `mowalisimulator_scheme-adapter_1` in the case of `/participants`, `/parties`, `/quotes`, and `/transfers` calls
   - `mowalisimulator_sim_1` in the case of `/reports` calls

The default port of the simulator is:

- `3000` in the case of `/participants`, `/parties`, `/quotes`, and `/transfers` calls
- `3002` in the case of `/reports` calls

Let's take an example `/parties` call:

```
curl -X GET http://<simulator IP address>:3000/parties/MSISDN/<test MSISDN> -H 'Accept:
application/vnd.interoperability.parties+json;version=1' -H 'Content-Type:
application/vnd.interoperability.parties+json;version=1.0' -H 'Date: Tue, 09 Apr 2019 10:13:38
GMT' -H 'FSPIOP-Source: <your fspId>'
```

## Testing as a Payee DFSP

To test being the Payee DFSP and receiving money, use the simulator Test API and the `POST /scenarios` request. This

will trigger an outbound transfer request to your DFSP implementation now acting as the Payee DFSP.

First, find out where the Test API should send the request to. When starting up the simulator, `docker-compose up` will generate some IP addresses on a local Docker network. To find out the IP address:

1. Find out the name of the simulator Docker network:
   - Execute `docker network ls` and look for the network with `mowali simulator` in its name.

     OR
   - Execute `docker-compose up` and note down the name of the network being created.
2. Execute: `docker network inspect <Docker network name>`
3. Look for the IPv4 address of the entry called `mowalisimulator_sim_1`. The default port to use is `3003`.

An example request that triggers and confirms an outgoing money transfer using Curl is provided below:

```
curl -X POST \
  http://<simulator IP address>:3003/scenarios \
  -H 'Content-Type: application/json' \
  -d '[
    {
        "name": "scenario1",
        "operation": "postTransfers",
        "body": {
            "from": {
                "displayName": "James Bush",
                "idType": "MSISDN",
                "idValue": "44123456789"
            },
            "to": {
                "idType": "MSISDN",
                "idValue": "44987654321"
            },
            "amountType": "SEND",
            "currency": "USD",
            "amount": "100",
            "transactionType": "TRANSFER",
            "note": "test payment",
            "homeTransactionId": "123ABC"
        }
    },
    {
        "name": "scenario2",
        "operation": "putTransfers",
        "params": {
            "transferId": "{{scenario1.result.transferId}}"
        },
        "body": {
            "acceptQuote": true
        }
    }
]'
```