sparts-docs Documentation

wind river

Nov 09, 2018

Contents

1 Introduction **3**

2 Short Description **5**
- 2.1 Why we need it . . . 6
- 2.2 Case Study . . . 6

3 Ledger **7**
- 3.1 Ledger Node Install & Launch Guide . . . 7
- 3.2 Ledger Node API . . . 10
- 3.3 Admin Guide . . . 27
- 3.4 Ledger Construction Guide . . . 27
- 3.5 Contributer Guide . . . 30

4 Command Line Interface **31**
- 4.1 User Guide . . . 31
- 4.2 Contributer Guide . . . 31

5 Atlas **33**
- 5.1 Why we need it . . . 33
- 5.2 How to register . . . 33

6 Frequently Asked Questions **35**
- 6.1 Endorsement . . . 35
- 6.2 Security & Access Control . . . 35

7 Community **37**
- 7.1 Discussion . . . 37
- 7.2 IRC chat . . . 37

Sparts allows you to track open source components used in your software supply chain. Keep track of an Open Source Bill of Matierals, cyrptography data, source code, legal notices, and other Compliance Artifacts. Before you ship production code, know that your suppliers have provided you with the proper Compliance Artifacts. Sparts is a decentralized immutable ledger built using Hyperledger Sawtooth.

CHAPTER 1

Introduction

CHAPTER 2

Short Description

Tracking all the different sub components used to construct a modern day device software can be challenging especially if components are contributed by multiple different suppliers. When we talk about devices think IoT things i.e., Internet of Things (IoT). The Software Parts initiative delivers a Sawtooth-based ledger that provides both access and accountability for relevant information for software parts exchanged among manufacturing supply chain participants. A software part is any software component that could be represented as one or more files. (e.g., binary library, source code package, application, container or an entire operating system runtime). Examples of the types of information tracked for a given software part include (but is not limited to) :

- **open source compliance artifacts** - The lion share of software today is comprised of some percentage of open source and therefore, legally, a software part needs to be accompanied by collection of required compliance artifacts (e.g., source code, notices, an open source bill of materials, SPDX documents and so forth). Providing access to and accountability over the required compliance artifacts is necessary to ensure one obtains the right to legally distribute their products. The ledger enables the tracking and assertion of *who* included *what* open source code, *how* and *when*.
- **certification evidence** - The objective of functional safety software is to create and present evidence that a software part has been certified (i.e., rigorously reviewed and tested) such that it mitigates unacceptable risk with respect to human physical injury or death. Providing access and accountability to the certification evidence is a necessary step in establishment trust among supply chain participants (e.g., autonomous vehicles, aircraft, medical devices, elevators, factory robots and so forth). The ledger enables the tracking and assertion of *who* included *what* evidence, *how* it was included and *when*.
- **cryptography usage** - Many governments (e.g., United States, France, UK, Russian, China to name a few) place restrictions of exporting software parts based on the implementation and/or usage of cryptography methods. Adhering to these restrictions and obtaining the appropriate export licenses is mission critical when exchanging software among international supply chain participants . The ledger enables the tracking and assertion of *who* included *what* cryptography code, *how* it was included and *when*.

2.1 Why we need it

2.2 Case Study

CHAPTER 3

Ledger

The Command Line Interace is an example of an application used to communicate with a ledger node. To setup a node follow the admin guide.

3.1 Ledger Node Install & Launch Guide

We discuss how to install a ledger node on different cloud platforms. The first platform we support and discuss is Amazon's Web Services (AWS). We are planning on providing instructions for Microsoft's Azure and Google Cloud platforms in the near future.

3.1.1 I) Installation on AWS

Configure the AWS instance

Go to the AWS EC2 dashboard and select any Linux based Amazon Machine Image. In this example we use Ubuntu Server 16.04.

Configure Network Ports

From the AWS console add the following **Inbound** rules for the instance's security group.

```
Port: 818               Destination: 0.0.0.0/0          Description: API
Port: 4004              Destination: 0.0.0.0/0          Description: Validator
Port: 22                Destination: 0.0.0.0/0          Description: SSH
```

Login into the ssh into your AWS instance (default user name for an Ubuntu Server is "ubuntu"). You can use ssh on linux and putty from windows.

Install Docker

Follow this guide to install docker (see Step 1): https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

Download and Run Initialize Script

With sudo privileges download and run the following scripts start-ledger:

```
git clone https://github.com/sparts-project/ledger-install-scripts.git
sparts-project/init-ledger.sh latest
```

The ledger container name is 'latest'

To test (ping) the ledger execute:

```
curl -i http://0.0.0.0:818/ledger/api/v1/ping
```

You can study the **init-ledger.sh** to understand the detailed steps of downloading and launching the ledger node container. You can use the **shutdown-ledger.sh** script to terminate container - warning: it will delete all the data and state information.

Initializing First User

You will need to add the first user (bootstrap) account. This needs to be done only once. You will need to specific:

- the public key, user account name (e.g., "johndoe")
- email address (e.g., john.doe@windriver.com);
- specific the authorization (e.g., "allow"); and
- the role (e.g., "admin").
- public key (e.g., "02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515")

```
sudo docker exec -ti latest sh -c "user register_init␣
↪02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515 johndoe john.
↪doe@windriver.com allow admin"
```

Additional Considerations

- Assigning an Elastic IP to instance
- AMI templating

3.1.2 II) Installation on Google Cloud

Go to the Google Cloud dashboard and select any Linux based Amazon Machine Image. In this example we use Ubuntu Server 16.04.

We recommend these minimum specifications for [TBA].

Create Firewall Rules

https://cloud.google.com/vpc/docs/using-firewalls

```
Go to cloud.google.com

Go to my Console

Choose you Project.

Choose Networking > VPC network

Choose "Firewalls rules"

Choose Create Firewall Rule

To apply the rule only to select VM instances, select Targets "Specified target tags",
↪ and enter into "Target tags" the tag which determine to which instances the rule␣
↪is applied. Then make sure the instances have the network tag applied. I have␣
↪created a target tag "ledger" and assigned it to the instance along with

select ingress rule

IP address ranges use: 0.0.0.0/0

To allow incoming TCP port 818, in "Protocols and Ports" enter tcp:818
To allow incoming TCP port 4004, in "Protocols and Ports" enter tcp:4004

Click Create
```

Install Docker

Follow this guide to install docker (see Step 1): https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

Download and Run Initialize Script

With sudo privileges download and run the following scripts start-ledger:

```
git clone https://github.com/sparts-project/ledger-install-scripts.git
sparts-project/init-ledger.sh latest
```

The ledger container name is 'latest'

To test (ping) the ledger execute:

```
curl -i http://0.0.0.0:818/ledger/api/v1/ping
```

You can study the **init-ledger.sh** to understand the detailed steps of downloading and launching the ledger node container. You can use the **shutdown-ledger.sh** script to terminate container - warning: it will delete all the data and state information.

Initializing First User

You will need to add the first user (bootstrap) account. This needs to be done only once. You will need to specific:

- the public key, user account name (e.g., "johndoe")

- email address (e.g., john.doe@windriver.com);
- specific the authorization (e.g., “allow”); and
- the role (e.g., “admin”).
- public key (e.g., “02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515”)

```
sudo docker exec -ti latest sh -c "user register_init␣
↪02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515 johndoe john.
↪doe@windriver.com allow admin"
```

Additional Considerations

3.2 Ledger Node API

3.2.1 I) Overview

The API for the SParts ledger is presented here. The ledger API calls are defined in part I of this document. The record types (objects) past between the ledger and client application are defined in part II of this document. Types include supplier, part, category as so forth.

3.2.2 II) Ledger API Calls

Ping Request

Send request to see if the ledger is currently available.

```
GET /ledger/api/v1/ping
```

Example of a successful response:

```
{   status:      "success",
    message:     "OK",
    result_type: "EmptyRecord",
    result:      {}
}
```

Since there is no data to return the record type **EmptyRecord** is specified in the results field. **EmptyRecord** is defined in part II of this document. If the ledger is not available then no response will be received.

Artifact Record

```
GET /ledger/api/v1/artifacts/{uuid}
```

(This call use to be: /api/ledger/envelopes/{uuid})

An artifact represents an item of evidence. Typically an artifact is a single document (e.g., notice file, source code archive, bill of materials). An envelope is a special instance of an artifact which represents a collection of artifacts potentially including other envelopes. For single artifacts the artifact_list field will be empty. For an envelope it will

contain a list of zero of more artifacts and the content_type field will be set to "envelope". The uri_list field is a list because copies of the artifact could exist in multiple locations.

Response form:

```
{   status:     "success",
    message:    "OK",
    result_type: "ArtifactRecord",
    result:     {
        name: "...",
        uuid: "...",
        filename: "...",
        checksum: "...",
        content_type: "...",  // envelope, notice, source, spdx, doc, other
        alias: "...",
        label: "...",
        openchain: "...",
        timestamp: "..."
        artifact_list: [...]   /* used for envelopes but not for singular artifact */
        uri_list: [ {
                    version: "...",
                             alias: "...",
                             checksum: "...",
                             size: "..."
                             content_type: "...",  // http, ipfs, ...
                             location: "https://...."
                           }
                         ]
                }
}
```

Example of a single artifact response:

```
{   status:     "success",
    message:    "OK",
    result_type: "ArtifactRecord",
    result: {
        name: "Zephyr 1.12 Notice File",
        uuid: "26559ed4-6868-488d-a5a7-3e81714beb00",
        filename: "Zephyr-1.12-Notices.txt",
        checksum: "f855d41c49e80b9d6f2a13148e5eb838607e92f1",
        content_type: "notices",
        alias: "zephyr-notices-1.12",
        label: "Zephyr Notices 1.12",
        openchain: "True",
        timestamp: "2018-06-18 00:30:12.498167"
        artifact_list: []   /* not used for singular artifact */
        uri_list: [ {
            version: "1.0",
               alias: "zephyr-notices-1.12",
               checksum: "Zephyr Notices 1.12",
               size:    "235120"
               content_type: "http",
               location: "https://...."
                   }
                 ]
}
```

Example of an envelope response:

```
{   status:      "success",
    message:     "OK",
    result_type: "ArtifactRecord",
    result: {     name: "Zephyr 1.12 Envelope",
                 uuid: "9b602058-c73f-4f02-9237-b71a2760fc15",
                 filename: "Zephyr-1.12-envelope.zip",
                 checksum: "a1e2486417f4cd7fc670bf5facd5870af9c1e3a5",
                 content_type: "envelope",
                 alias: "zephyr-notices-1.12",
                 label: "Zephyr Notices 1.12",
                 openchain: "True",
                 timestamp: "2018-06-18 00:30:12.498167"
                 artifact_list: [
                              { uuid: "731ef148-5f81-11e8-9c2d-fa7ae01bbebc",
                                 path: "/spdx"},
                             { uuid: "f2cef148-5f81-11e8-8f51-fa7ae01bb93b",
                               path: "/notices"}
                 ]
                 uri_list: [ {
                                  version: "1.0",
                                 alias: "zephyr-envelope-1.12",
                                 checksum: "f67d3213907a52012a4367d8ad4f093b65abc016",
                                 size:    "235120"
                                 content_type: "http",
                                 location: "https://...."
                             }
                           ]
}
```

Note that the envelope record utilizes the artifact_list field where a single artifact does not.

Artifact Add

```
POST /ledger/api/v1/artifacts
```

Use the **ArtifactRecord** (the artifact_list and uri_list fields are not used in this post). The request must be performed by a user with Roles: admin or supplier.

| Field | Type | Description | | ——————— | ——— | ——————————————————————— | | uuid | string | unique identifier | | name | string | file or envelope name | | alias (was short_id) | string | alias for typing | | label | string | // Display name | | checksum | string | artifact checksum | | openchain | string | true/false If prepared under an OpenChain comforting program | | content_type | string | envelope, notices, spdx, source, . . . |

An example single artifact request:

```
{
   private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   artifact: {
               uuid: "7709ca8d-01f4-4de2-69ed-16b7ebae704a",
               name: "Zephypr 1.12 SPDX file",
               alias: "zephypr_1.12",
               label: "Zephypr 1.12 SPDX file",
               checksum: "f855d41c49e80b9d6f2a13148e5eb838607e92f1",
               openchain: true,
               content_type: "spdx"
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Example curl Request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K92SiHianMJRtqRiMaQ6xwzuYz7xaFRa2C8ruBQT6edSBg87Kq", "public_key" :
↪"02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515", "artifact": {
↪"uuid": "7709ca8d-01f4-4de2-69ed-16b7ebae705c","name": "Zephypr 1.12 SPDX file",
↪"checksum": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "alias": "zephypr_1.12",
↪"label": "Zephypr 1.12 SPDX file", "openchain": "true", "content_type": "spdx"} }'␣
↪http://147.11.176.111:818/ledger/api/v1/artifacts
```

Potential Errors:

- The requesting user does not have the appropriate access credentials to perform the add.
- One or more of the required fields UUID, checksum are missing.
- The UUID is not in a valid format.

Artifact URI Add

```
POST /ledger/api/v1/artifacts/uri
```

The request must be performed by a user with Roles: Admin or Supplier.

| Field | Type | Description | | ———— | —— | ——————————— | | version | string | name of use | | checksum | string | artifact checksum | | content_type | string | type (e.g., text, binary, archive, other) | | size | int | file size in bytes | | uri_type | string | (e.g., http, ipfs, . . .) | | location | string | link, path |

An example request:

```
{  private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   uuid: "bcb083a1-89c7-4bd2-a568-8450350e8195",
   uri:  { version: "1.0",
           checksum: "f67d3213907a52012a4367d8ad4f093b65abc016",
           size:    "235120"
           content_type: ".pdf",
           uri_type: "http",
           location:
              "https://github.com/zephyrstorage/_content/master/
↪f67d3213907a52012a4367d8abc016"
        }
}
```

The uri field is of type **URIRecord**.

Example curl request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K92SiHianMJRtqRiMaQ6xwzuYz7xaFRa2C8ruBQT6edSBg87Kq", "public_key" :
↪"02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515",
"uuid": "7709ca8d-01f4-4de2-69ed-16b7ebae705c", "uri": {"version": "1.0", "checksum":
↪"f855d41c49e80b9d6f2a13148e5eb838607e92f1", "size": "235120", "content_type": ".pdf
↪", "uri_type": "http", "location": "https://github.com/zephyrstorage/_content/
↪master/f67d3213907a52012a4367d8abc016" } }' http://147.11.176.111:818/ledger/api/v1/
↪artifacts/uri
```

(continues on next page)

(continued from previous page)

Artifact Of Envelope Relation

```
POST /ledger/api/v1/envelope/artifact
```

The user is identified by the public key. The following input record is: **EnvelopeArtifactRecord**

```
{  private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   relation: {
                artifact_uuid: "f855d41c49e80b9d6f2a13148e5eb838607e92f1",
                envelope_uuid: "dec6b86a-f794-43d6-64bc-ca4146548048"
            }
}
```

Example curl request

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt", "public_key" :
↪"034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9", "relation": {
↪"artifact_uuid": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "envelope_uuid":
↪"dec6b86a-f794-43d6-64bc-ca4146548048"} }' http://localhost:3075/ledger/api/v1/
↪envelope/artifact
```

To do:

- Ledger check for envelope to be an envelope or send back error
- Need to use uuid for user. - are public keys unique??

POST /ledger/api/v1/relation/part_artifact POST /ledger/api/v1/relation/category_part

Artifact of Part Relation

```
POST /ledger/api/v1/artifacts/part
```

ArtifactOfPartRecord

Example request:

```
{  private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   relation: {
                part_uuid: "cb00a696-14d0-447a-6096-69be4c5d93a5",
                artifact_uuid: "2745a756-eed8-4093-683f-a1f6b56f7249"
            }
}
```

Example curl request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt", "public_key" :
↪"034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9", "relation": {
↪"part_uuid": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "artifact_uuid": "dec6b86a-
↪f794-43d6-64bc-ca4146548048"} }' http://localhost:3075/ledger/api/v1/artifacts/part
```

Organization

A organization can represent a company, foundation, project or individual. The record fields include:

- **uuid** [string] - universal unique identifier
- **name** [string] - Name of customer
- **alias** [string] - short identifier 1-15 alphanumeric characters
- **type** [string] - optional - include multiple simple (1-3) word type descriptions each separated by ';' to describe the organization's type. For example, "supplier; ledger host"
- **description** [string] - (optional) - brief 1-3 sentence description
- **url** [string] - (optional) - Organization's web homepage url (if one exists)

Organization Record

Retrieve the record of an organization:

```
GET /ledger/api/v1/orgs/{uuid}
```

General Response:

```
{   status:      "success",
    message:     "OK",
    result_type: "OrganizationRecord",
    result:      {   name: "...",
                     uuid: "...",
                     alias: "...",
                     type: "..."
                     description: "...",
                     url: "..."
                 }
}
```

Example Response:

```
{   status:      "success",
    message:     "OK",
    result_type: "ORecord",
    result: {   name: "Tesla, Inc.",
                uuid: "31e3e600-cd79-4ee5-464e-e74e1ce763cc",
                alias: "Tesla",
                type: "supplier",
                description: "Company specializing in electric vehicles and
                              lithium-ion battery energy storage.",
                url: "http://www.tesla.com"
            }
}
```

If an organization record is not found the following error response will be received:

```
{   status:      "failed",
    message:     "Organization record not found",
    result_type: "EmptyRecord",
    result:      {}
}
```

Organization List

Obtain a list of organizations.

```
GET /ledger/api/v1/orgs
```

Example Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "ListOf:OrganizationRecord",
    result: [ { name: "Tesla, Inc.",
                uuid: "31e3e600-cd79-4ee5-464e-e74e1ce763cc",
                alias: "Tesla",
                type: "customer",
                description: "Company specializing in electric vehicles and
                              lithium-ion battery energy storage."
                url: "http://www.tesla.com"
            },
            {   name: "General Motors Corporation",
                uuid: ""2584a6ce-16a7-44c0-7e53-21969d1e026b",
                alias: "GM",
                type: "customer"
                description: "United States automotive manufacturer."
                url: "http://www.gm.com"
            },
            {   name: "Wind River Systems",
                uuid: ""3568f20a-8faa-430e-7c65-e9fce9aa155d",
                alias: "WindRiver",
                type: "supplier"
                description: "United States automotive manufacturer."
                url: "http://www.windriver.com"
            }
    ]
}
```

If there are no organizations registered then the empty list will be returned:

Empty List Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "ListOf:OrganizationRecord",
    result: [ ]
}
```

Organization Add

```
POST /ledger/api/v1/orgs
```

Add a organization entity to the ledger (e.g., supplier, customer, ledger host). The request must be performed by a user with Role:Admin access. The POST parameters are:

- **uuid** [string] - universal unique identifier
- **name** [string] - Name of customer
- **alias** [string] - short identifier 1-15 alphanumeric characters (evaluated as case insensitive)

- **type** [string] - optional - include multiple simple (1-3) word type descriptions each separated by ';' to describe the organization's type. For example, "supplier; ledger host"
- **description** [string] - (optional) - brief 1-3 sentence description
- **url** [string] - (optional) - Organization's web homepage url (if one exists)

Example Request:

```
{
    uuid: "31e3e600-cd79-4ee5-464e-e74e1ce763cc",
    name: "Tesla, Inc.",
    alias: "Tesla",
    description: "Company specializing in electric vehicles and
                  lithium-ion battery energy storage."
    url: "http://www.windriver.com"
}
```

Example curl command:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"name": "Wind River", "uuid
↪": "3568f20a-8faa-430e-7c65-e9fce9aa155d", "alias": "WindRiver", "url": "http://www.
↪windriver.com"}' http://localhost:818/ledger/api/v1/orgs
```

Supplier

Supplier Record

```
GET /ledger/api/v1/suppliers/{uuid}
```

Response:

```
{   status:      "success",
    message:     "OK",
    result_type: "SupplierRecord",
    result:      {    name: "...",
                      uuid: "...",
                      alias: "...",
                      url: "...",
                      parts: [...]
                 }
}
```

Example Response:

```
{   status:      "success",
    message:     "OK",
    result_type: "SupplierRecord",
    result: {    name: "Wind River Systems",
                 uuid: "dde3e600-cd79-4ee5-464e-e74e1ce764bb",
                 alias: "WindRiver",
                 parts: [
                      { part_id: "731ef148-5f81-11e8-9c2d-fa7ae01bbebc" },
                      { part_id: "f2cef148-5f81-11e8-8f51-fa7ae01bb93b" },
                      { part_id: "ee3b9d57-4c98-4d6f-5ecb-a54c97a7cda2" }
                 ],
```

(continues on next page)

(continued from previous page)

```
                url: "http://www.windriver.com"
            }
}
```

See section II for the json definition of **OrganizationRecord**. If an organization is not found the following error response will be received:

```
{   status:     "failed",
    message:    "Organization record not found",
    result_type: "EmptyRecord",
    result:     {}
}
```

Supplier List

Obtain a list of suppliers.

```
GET /ledger/api/v1/suppliers
```

Example Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "ListOf:SupplierRecord",
    result: [ { name: "Wind River Systems",
                uuid: "dde3e600-cd79-4ee5-464e-e74e1ce764bb",
                alias: "WindRiver",
                parts: [
                    { part_uuid: "731ef148-5f81-11e8-9c2d-fa7ae01bbebc" },
                    { part_uuid: "f2cef148-5f81-11e8-8f51-fa7ae01bb93b" },
                    { part_uuid: "ee3b9d57-4c98-4d6f-5ecb-a54c97a7cda2" }
                ],
                url: "http://www.windriver.com"
            },
            {   name: "Intel Corporation",
                uuid: ""2584a6ce-16a7-44c0-7e53-21969d1e026b",
                alias: "Intel",
                parts: [
                    { part_uuid: "6584a6ce-16a7-44c0-7e53-21969d1e026b" },
                ],
                url: "http://www.intel.com"
            }
    ]
}
```

See section II for the json definition of **SupplierRecord**. If there are not organizations registered then the empty list will be received:

```
{   status:     "success",
    message:    "OK",
    result_type: "ListOf.SupplierRecord",
    result: [ ]
}
```

Supplier Add

```
POST /ledger/api/v1/suppliers
```

Add a supplier entity to the ledger. The request must be performed by a user with Role:Admin access. The POST parameters are:

- **uuid** [string] - universal unique identifier
- **name** [string] - Name of customer
- **alias** [string] - short identifier 1-15 alphanumeric characters
- **description** [string] - (optional) - brief 1-3 sentence description
- **url** [string] - (optional) - web home url

| Name | Type | Description | | —— | ——- | ———————————————— | | uuid | string | universal unique identifier | | name | string | name of organization | | alias | strings | quick identifier 1-15 alphanumeric characters website | | url | string | website url |

Example Request:

```
{
    uuid: "dde3e600-cd79-4ee5-464e-e74e1ce764bb",
    name: "WindRiver",
    alias: "supplier-76bb1",
    url: "http://www.windriver.com"
}
```

Example curl command:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"name": "Wind River", "uuid
↪": "3568f20a-8faa-430e-7c65-e9fce9aa155d", "alias": "wr", "url": "http://www.
↪windriver.com"}' http://localhost:3075/ledger/api/v1/orgs
```

Register User

```
POST /ledger/api/v1/registeruser
```

Use the **UserRegisterRecord**. The request must be performed by a user with Roles: admin or supplier.

| Field | Type | Description | | ———— | ———- | ————— | | private_key | string | unique identifier | | public_key | string | file or envelope name | | user | UserRecord | alias for typing |

With **UserRecord**:

| Field | Type | Description | | ————- | —— | ———————— | | user_name | string | user name | | email_address | string | user email | | role | string | the role (e.g., “member”) | | authorized | string | the authorization (e.g., “allow”) | | public_key | string | the user’s pu |

An example single artifact request:

```
{
   private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   user: {
            user_name: "7709ca8d-01f4-4de2-69ed-16b7ebae704a",
```

(continues on next page)

(continued from previous page)

```
            email_address: "John.Doe@intel.com",
            role: "member",
            authorized: "allow",
            public_key:
                "03ef24753779355b4841dcef68a28044d1bc41b508b75bf8455b8518a5a61da50a"
        }
}
```

Example curl Request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K92SiHianMJRtqRiMaQ6xwzuYz7xaFRa2C8ruBQT6edSBg87Kq", "public_key" :
↪"02be88bd24003b714a731566e45d24bf68f89ede629ae6f0aa5ce33baddc2a0515", "artifact": {
↪"uuid": "7709ca8d-01f4-4de2-69ed-16b7ebae705c","name": "Zephypr 1.12 SPDX file",
↪"checksum": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "alias": "zephypr_1.12",
↪"label": "Zephypr 1.12 SPDX file", "openchain": "true", "content_type": "spdx"} }'␣
↪http://147.11.176.111:818/ledger/api/v1/artifacts
```

Potential Errors:

- The requesting user does not have the appropriate access credentials to perform the add.
- One or more of the required fields UUID, checksum are missing.
- The UUID is not in a valid format.

Part Get

To obtain a part record use:

```
GET /ledger/api/v1/parts/{uuid}
```

Response Record

```
{   status:     "success",
    message:    "OK",
    result_type: "PartRecord",
    result: {    uuid: "...",
                name: "...",
                version: "...",
                label: "..."
                alias: "..."
                checksum: "...",
                label: "...",
                licensing: "..."
                description: "..."
                artifacts: [...]
                suppliers: [...]
                categories: [...]
    }
}
```

Example response:

```
{   status:     "success",
    message:    "OK",
```

(continues on next page)

(continued from previous page)

```
    result_type: "PartRecord",
    result: {    uuid: "f1eae70d-86ba-4440-583a-28127e447f83",
                name: "Zephyr Runtime 1.10",
                version: "1.10",
                label: "zephyr 1.10"
                alias: "z1
                description: "Zephyr runtime for the ACX 11 board support bpackage"
                licensing: "Apache-2.0",
                checksum: "d9be5fcf820e88b217a760f7869959af49898dbe",
                suppliers: [
                              { supplier_id: "3568f20a-8faa-430e-7c65-e9fce9aa155d" }
                           ],
                artifacts: [ ],
                categories: [ ],
    }
```

Error Messages include:

- Part uuid not found

Part List

```
GET /ledger/api/v1/parts
```

Returns a list of **PartRecords**. Example response:

```
{   status:      "success",
    message:     "OK",
    result_type: "ListOf.PartRecord",
    result: []
}
```

Part Add

```
POST /ledger/api/v1/parts
```

Example:

```
{  private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   part: {
             uuid: "731ef148-5f81-11e8-9c2d-fa7ae01bbebc",
             name: "Zephyr Runtime 1.10",
             label: "Zephyr 1.10"
             alias: "zephyr_1.10"
             version: "1.10",
             checksum: "1e673213907a52012a4367d8ad4f093b65abc222",
             label: "Test Part 1.0",
             licensing: "MIT"
             description: "Zephyr is a small real-time operating supporting multiple␣
↪architectures"
          }
}
```

Example curl request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt", "public_key" :
↪"034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9", "part": {"uuid
↪": "731ef148-5f81-11e8-9c2d-fa7ae01bbebc","name": "Zephypr 1.12 SPDX file", "version
↪": "1.12", "checksum": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "alias":
↪"zephypr_1.10", "label": "Zephyr 1.10", "licensing": "MIT", "description": "Zephyr␣
↪is a small real-time operating supporting multiple architectures" } }' http://
↪localhost:3075/ledger/api/v1/parts
```

Part Of Supplier Relation

```
POST /ledger/api/v1/parts/supplier
```

PartOfSupplierRecord

Example request:

```
{  private_key: "5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt",
   public_key: "034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9",
   relation: {
                supplier_uuid: "cb00a696-14d0-447a-6096-69be4c5d93a5",
                part_uuid: "2745a756-eed8-4093-683f-a1f6b56f7249"
            }
}
```

Example curl request:

```
curl -i -H "Content-Type: application/json" -X POST -d  '{"private_key":
↪"5K9ft3F4CDHMdGbeUZSyt77b1TJavfR7CAEgDZ7nXbdno1aynbt", "public_key" :
↪"034408551a7b24b917103ccfafb402195713cd2e5dcdc588e7dc537f07b195bcf9", "relation": {
↪"part_uuid": "f855d41c49e80b9d6f2a13148e5eb838607e92f1", "supplier_uuid": "dec6b86a-
↪f794-43d6-64bc-ca4146548048"} }' http://localhost:3075/ledger/api/v1/parts/supplier
```

Get Public/Private Key Pair

```
GET /ledger/api/v1/keys
```

Example Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "PrivatePublicKeyRecord",
    result: {
              private_key: "5K6Q2kMHaMUrRjvSb4EPXQEJi1nAy3uXAYhqYvq2qNiLEGuFuVS",
              public_key:
↪"0315d60b8dd9a90c55f2f7643270bc46d20798c1e5a38a30c9cb839882398d537f"
    }
}
```

User Add

```
POST /ledger/api/v1/users
```

Here is an example of how to register a new user.

The request must be performed by a user with Role:Admin.

| Name | Type | Description | | ———— | —— | ——————————- | | name | string | name of user | | email | string | The email address of user | | role | string | specific role value [admin, supplier] | | authorized | string | specific access value [allow, deny] | | public_key | string | public key value |

Example Request:

```
{   name:   "John Doe",
    email:  "john.doe@windriver.com",
    role: "admin",
    authorized: "allow",
    public_key: "0315d60b8dd9a90c55f2f7643270bc46d20798c1e5a38a30c9cb839882398d537f"
}
```

Example response:

```
{   status:     "success",
    message:    "OK",
    result_type: "EmptyRecord",
    result: {}
}
```

Example curl request:

```
 curl -i -H "Content-Type: application/json" -X POST -d  '{"name": "John Doe", "email
↪": "john.doe@windriver.com", "role": "admin", "authorized": "allow", "public_key":
↪"0315d60b8dd9a90c55f2f7643270bc46d20798c1e5a38a30c9cb839882398d537f"}' http://
↪localhost:3075/ledger/api/v1/users
```

User Get

```
GET /ledger/api/v1/users/{public_key}
```

```
{   name:   "Sameer Ahmed",
    email:  "sameer.ahmed@windriver.com",
    organization: "Wind River"
    public_key: "0315d60b8dd9a90c55f2f7643270bc46d20798c1e5a38a30c9cb839882398d537f"
}
```

GET Category

```
GET /ledger/api/v1/categories/{uuid}
```

Example Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "CategoryRecord",
    result: {
        name: "OS",
        description: "Operating System",
        uuid: "43903f02-00fd-43a3-bdaa-befe4a2fcd7e"
    }
}
```

An example error response where the category uuid does not exist:

```
{   status:     "failed",
    message:    "Category record not found",
    result_type: "EmptyRecord",
    result:     {}
}
```

List Categories

Obtain a list of the categories.

```
GET /ledger/api/v1/categories
```

Example Response:

```
{   status:     "success",
    message:    "OK",
    result_type: "ListOf.CategoryRecord",
    result: [ { name: "operating system",
                description: "Operating System",
                uuid: "43903f02-00fd-43a3-bdaa-befe4a2fcd7e"
            },
            {   name: "libraries",
                description: "Operating System",
                uuid: "12d7f0c2-00fd-43a3-bdaa-befe4a2fcd7e"
            }
    ]
}
```

3.2.3 III) API Types

ArtifactRecord

```
{
    UUID          string          `json:"uuid"`
    Name          string          `json:"name"`
```

(continues on next page)

(continued from previous page)

```
    Alias        string         `json:"short_id,omitempty"`
    Label        string         `json:"label,omitempty"`  // Display name
    Checksum     string         `json:"checksum"`
    OpenChain    string         `json:"openchain,omitempty"`
    ContentType  string         `json:"content_type,omitempty"`
    Timestamp    string         `json:"timestamp,omitempty"`
    ArtifactList ListOf.ArtifactItem `json:"artifact_list,omitempty"`
    URIList      ListOf.URIRecord    `json:"uri_list, omitempty"`
}
```

ArtifactItem

```
{
    UUID string `json:"uuid"` // Artifact Universal Unique IDentifier
    Path string `json:"path"` // Path of artifact within the envelope
}
```

EmptyRecord

```
{ }
```

EnvelopeArtifactRecord

```

```

PartItemRecord

```
{
    PartUUID string `json:"part_uuid"` // Part uuid
}
```

PartRecord

```
{
    Name        string `json:"name"`                  // Fullname
    Version     string `json:"version,omitempty"`     // Version if exists.
    Alias       string `json:"label,omitempty"`       // 1-15 alphanumeric characters
    Licensing   string `json:"licensing,omitempty"`   // License expression
    Description string `json:"description,omitempty"` // Part description (1-3␣
↪sentences)
    Checksum    string `json:"checksum,omitempty"`    // License expression
    UUID        string `json:"uuid"`                  // UUID provide w/previous␣
↪registration
    URIList     []URIRecord `json:"uri_list,omitempty"`     //
}
```

PrivatePublicKeyRecord

```
{
    PrivateKey string `json:"private_key"` // Private key
    PublicKey  string `json:"public_key"`  // Pubkic key
}
```

UserRecord

```
{
    Name       string `json:"user_name"`
    Email      string `json:"email_address"`
    Role       string `json:"role"`
    Authorized string `json:"authorized"`
    PublicKey  string `json:"user_public_key"`
}
```

UserRegisterRecord

```
{
    User       UserRecord `json:"user"`
    PrivateKey string     `json:"private_key"`
    PublicKey  string     `json:"public_key"`
}
```

SupplierRecord

```
{
    UUID    string `json:"uuid"`               // universal unique identifier
    Name    string `json:"name"`               // Fullname
    Alias   string `json:"alias"`    // 1-15 alphanumeric characters
    Url     string `json:"url"`      // 2-3 sentence description
    Parts   ListOf:PartItemRecord
}
```

URIRecord

```
{
    Version     string `json:"version"`
    Checksum    string `json:"checksum"`
    ContentType string `json:"content_type"`   // text, envelope, binary, archive
    Size        string `json:"size,omitempty"` // size in bytes
    URIType     string `json:"uri_type"`       // e.g., http, ipfs
    Location    string `json:"location"`       // actual link
}
```

3.3 Admin Guide

3.3.1 Installation

3.3.2 Configuring Webserver

3.4 Ledger Construction Guide

These instructions will get you an instance of a ledger up and running on your local machine for development and testing purposes.

3.4.1 Installing Docker

Installation Tutorial

The Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version. To get this latest version, install Docker from the official Docker repository. This section shows is quoted from the following tutorial:

(Step 1) https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

First, in order to ensure the downloads are valid, add the GPG key for the official Docker repository to your system:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable"
```

Next, update the package database with the Docker packages from the newly added repo:

```
$ sudo apt-get update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:

```
$ apt-cache policy docker-ce
```

You should see output similar to the follow:

```
docker-ce:
  Installed: (none)
  Candidate: 18.06.1~ce~3-0~ubuntu
  Version table:
     18.06.1~ce~3-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
```

Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 16.04 (xenial).

Finally, install Docker:

```
$ sudo apt-get install -y docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ sudo systemctl status docker
```

The output should be similar to the following, showing that the service is active and running:

```
 docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:␣
↪enabled)
   Active: active (running) since Thu 2018-10-18 20:28:23 UTC; 35s ago
     Docs: https://docs.docker.com
 Main PID: 13412 (dockerd)
   CGroup: /system.slice/docker.service
           ├─13412 /usr/bin/dockerd -H fd://
           └─13421 docker-containerd --config /var/run/docker/containerd/containerd.
↪toml
```

Check whether Docker is working correctly and that you have access to Docker Hub using the command:

```
$ docker run hello-world
```

Output:

```
...
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

3.4.2 Creating a container within Docker with Ubuntu 16.04

With Docker installed, we can now create a container with Ubuntu 16.04.

Locating and Loading a Ubuntu image

To pull the Ubuntu image from DockerHub, use the command:

```
$ docker pull ubuntu
```

Spinning up and executing a container (needs revision)

To spin up a container named `$CONTAINER`, use `docker run`:

```
sudo docker run -dit --name=$CONTAINER -p 0.0.0.0:818:818 -p 0.0.0.0:4004:4004 -p 127.
↪0.0.1:8080:8080 -p 127.0.0.1:8800:8800 ubuntu:$CONTAINER
```

3.4.3 Installing Sawtooth v.1.0.5 (add comments)

In this section, we discuss the installation of Sawtooth on top of a local ubuntu container. Tutorial

Getting the Sawtooth packages for Ubuntu (add dependencies)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪8AA7AF1F1091A5FD
$ sudo add-apt-repository 'deb http://repo.sawtooth.me/ubuntu/1.0/stable xenial␣
↪universe'
$ sudo apt-get update
```

Install Sawtooth

```
$ sudo apt-get install -y sawtooth
```

Create the genesis block

```
$ sawtooth keygen
$ sawset genesis
$ sudo -u sawtooth sawadm genesis config-genesis.batch
```

Output:

```
Processing config-genesis.batch...
Generating /var/lib/sawtooth/genesis.batch
```

3.4.4 Configure Sawtooth Validator and REST API

Sawtooth validator

To start a validator that listens locally on the default ports, run the following commands:

```
$ sudo sawadm keygen
$ sudo -u sawtooth sawtooth-validator -vv
```

Logging output from the validator should look similar to:

```
[2017-12-05 22:33:42.785 INFO     chain] Chain controller initialized with chain␣
↪head: c788bbaf(2, S:3073f964, P:c37b0b9a)
[2017-12-05 22:33:42.785 INFO     publisher] Now building on top of block: c788bbaf(2,
↪ S:3073f964, P:c37b0b9a)
[2017-12-05 22:33:42.788 DEBUG    publisher] Loaded batch injectors: []
[2017-12-05 22:33:42.866 DEBUG    interconnect] ServerThread receiving TP_REGISTER_
↪REQUEST message: 92 bytes
[2017-12-05 22:33:42.866 DEBUG    interconnect] ServerThread receiving TP_REGISTER_
↪REQUEST message: 103 bytes
[2017-12-05 22:33:42.867 INFO     processor_handlers] registered transaction␣
↪processor: connection_
↪id=4c2d581131c7a5213b4e4da63180048ffd8983f6aa82a380ca28507bd3a96d40027a797c2ee59d029e42b7b1b4cc4706
↪ family=intkey, version=1.0, namespaces=['1cf126']
[2017-12-05 22:33:42.867 DEBUG    interconnect] ServerThread sending TP_REGISTER_
↪RESPONSE to b'c61272152064480f'
```

(continues on next page)

(continued from previous page)

```
[2017-12-05 22:33:42.869 INFO     processor_handlers] registered transaction␣
↪processor: connection_
↪id=e80eb89943398f296b1c99e45b5b31a9647d1c15a412842c804222dcc0e3f3a3045b6947bab06f42c5f79acdcde91be4
↪ family=sawtooth_settings, version=1.0, namespaces=['000000']
[2017-12-05 22:33:42.869 DEBUG    interconnect] ServerThread sending TP_REGISTER_
↪RESPONSE to b'a85335fced9b496e'
```

Sawtooth REST API

In order to configure a running validator, submit batches, and query the state of the ledger, you must start the REST API application. Connect to the validator via the following command:

```
$ sudo -u sawtooth sawtooth-rest-api -v
```

3.4.5 Startup SParts Transaction Processors

In progress.

3.4.6 License

This project is licensed under **[[Insert License Here]]** - see the LICENSE.md file for details

3.5 Contributer Guide

3.5.1 Project Structure

3.5.2 Example Pull Request

CHAPTER 4

Command Line Interface

The Command Line Interace is an example of an application used to communicate with a ledger node. You can download the CLI from this github repo

4.1 User Guide

4.1.1 Basic Commands

4.1.2 Advanced Commands

4.1.3 Troubleshooting

4.2 Contributer Guide

4.2.1 Project Structure

4.2.2 Example Pull Request

CHAPTER 5

Atlas

5.1 Why we need it

5.2 How to register

CHAPTER 6

Frequently Asked Questions

6.1 Endorsement

Endorsement architecture:

Question How many peers in the network need to endorse a transaction?

Answer The number of peers required to endorse a transaction is driven by the endorsement policy that is specified at chaincode deployment time.

6.2 Security & Access Control

Question How do I ensure data privacy?

Answer There are various aspects to data privacy.

CHAPTER 7

Community

7.1 Discussion

7.2 IRC chat