FAIR Data Point

Release 1.8.0

Dutch Techcentre for Life Sciences

Mar 10, 2021

ABOUT

1 About FAIR Data Point **1**
- 1.1 Features . . . 1
- 1.2 Security . . . 1

2 Users and Roles **3**
- 2.1 FAIR Data Point Roles . . . 3
- 2.2 Catalog Roles . . . 3
- 2.3 Dataset Roles . . . 3

3 Components **5**
- 3.1 Triple Store . . . 6
- 3.2 MongoDB . . . 6
- 3.3 FAIRDataPoint . . . 6
- 3.4 FAIRDataPoint-client . . . 6
- 3.5 Reverse Proxy . . . 6

4 Local Deployment **7**
- 4.1 Running locally on a different port . . . 8
- 4.2 Persistence . . . 8

5 Production Deployment **11**

6 Advanced Configuration **15**
- 6.1 Triple Stores . . . 15
- 6.2 Mongo DB . . . 17
- 6.3 Default attached metadata . . . 17
- 6.4 Customizations . . . 17
- 6.5 Running FDP on a nested route . . . 18

7 Usage **21**
- 7.1 Resource definitions . . . 21
- 7.2 Shapes . . . 22

8 API Usage **25**
- 8.1 Obtaining an API token . . . 25
- 8.2 Interacting with metadata . . . 25
- 8.3 API endpoint listing . . . 27

9 Usage **29**
- 9.1 About metadata extension . . . 29
- 9.2 Features . . . 29

10 Setup **31**
10.1 Installation 31
10.2 Configuration 32
10.3 Compatibility 33

11 Contributing **35**
11.1 Development 35

12 Changelog **37**
12.1 Overview 37
12.2 Detailed changelog 38

CHAPTER

ONE

ABOUT FAIR DATA POINT

FAIRDataPoint is a REST API and Web Client for creating, storing, and serving **FAIR metadata**. The metadata contents are generated **semi-automatically** according to the FAIR Data Point software specification document.

1.1 Features

- Store catalogs, datasets, and distributions
- Manage users
- Manage access rights to your catalogs, datasets, and distributions

1.2 Security

We have two levels of accessibility in FDP. All resources (e.g., catalogs, datasets,...) are publicly accessible. You don't need to be logged in to browse them. If you want to upload your own resources, you need to be logged in. To get an account, you need to contact an administrator of the FDP. By default, all uploaded resources are publicly accessible by anyone. But if you want to allow someone to manage your resources (edit/delete), you need to allow it in the resource settings.

We have two types of roles in FDP - an administrator and a user. The administrator is allowed to manage users and all resources. The user can manage just the resources which he owns.

CHAPTER

TWO

USERS AND ROLES

There are different roles for different levels in the FAIR Data Point.

2.1 FAIR Data Point Roles

2.1.1 Admin

Admin can manage other user accounts and access everything in the FAIR Data Point.

2.1.2 User

User can create new catalogs and access existing catalogs where she was added.

2.2 Catalog Roles

2.2.1 Owner

Owner can update catalog details, add other users and upload new datasets.

2.2.2 Data Provider

Data Provider can create new data sets in the catalog.

2.3 Dataset Roles

2.3.1 Owner

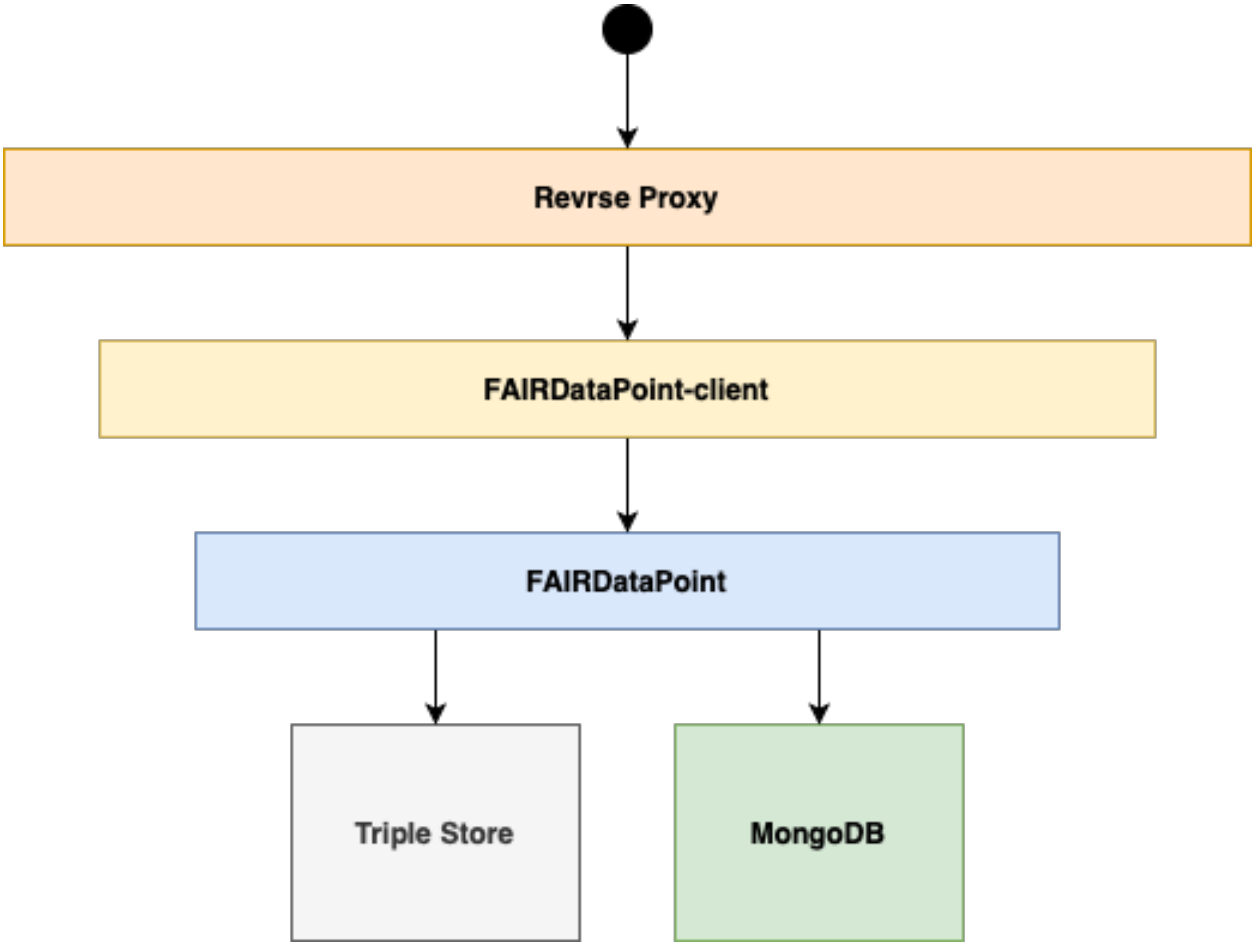
Owner of the data set can update catalog details and add other users.

CHAPTER

THREE

COMPONENTS

The deployment of the FAIR Data Point consists of a couple of components. See the following image for the overview:



3.1 Triple Store

Every FAIR Data Point needs to store the semantic data somewhere. A triple Store is a place where the data is. It is possible to configure different stores, see *Triple Stores configuration* for more details.

3.2 MongoDB

Besides semantic data, FDP needs information about user accounts and their roles. These data are stored in MongoDB database.

3.3 FAIRDataPoint

FAIRDataPoint is distributed in Docker image `fairdata/fairdatapoint`. It is the core component which handles all the business logic and operations with the semantic data. It also provides API for working with data in different formats.

3.4 FAIRDataPoint-client

FDP client is distributed in Docker image `fairdata/fairdatapoint-client`. It provides the user interface for humans. It works as a reverse proxy in front of the FAIR Data Point which decides whether the request is for machine-readable data and passes it to the FAIRDataPoint or from a web browser in which case it serves the interface for humans.

3.5 Reverse Proxy

In a production deployment, there is usually a reverse proxy that handles HTTPS certificates, so the connection to the FAIR Data Point is secured. See *production deployment* to learn how to configure one.

CHAPTER

FOUR

LOCAL DEPLOYMENT

FAIR Data Point is distributed in Docker images. For a simple local deployment, you need to run `fairdatapoint`, `fairdatapoint-client` and `mongo` images. See the *Components* section to read more about what each image is for.

Here is an example of the simplest Docker Compose configuration to run FDP.

```
# docker-compose.yml

version: '3'
services:

    fdp:
        image: fairdata/fairdatapoint:1.8.0

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        ports:
            - 80:80
        environment:
            - FDP_HOST=fdp

    mongo:
        image: mongo:4.0.12
```

Then you can run it using `docker-compose up -d`. It might take a while to start. You can run `docker-compose logs -f` to follow the output log. Once you see a message, that the application started, the FAIR Data Point should be working, and you can open http://localhost.

There are two default user accounts. See the *Users and Roles* section to read more about users and roles. The default accounts are

User name	Role	Password
albert.einstein@example.com	admin	password
nikola.tesla@example.com	user	password

Danger: Using the default accounts is alright if you run FDP on your machine, but you should change them if you want to run FDP publicly.

4.1 Running locally on a different port

If you want to run the FAIR Data Point locally on a different port than the default `80`, additional configuration is necessary. First, we need to create a new file `application.yml` and set the client URL to the actual URL we want to use.

```
# application.yml

instance:
    clientUrl: http://localhost:8080
```

Then, we need to mount the application config into the FDP container and update the port which the FDP client runs on.

```
# docker-compose.yml

version: '3'
services:

    fdp:
        image: fairdata/fairdatapoint:1.8.0
        volumes:
            - ./application.yml:/fdp/application.yml:ro

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        ports:
            - 8080:80
        environment:
            - FDP_HOST=fdp

    mongo:
        image: mongo:4.0.12
```

4.2 Persistence

We don't have any data persistence with the previous configuration. Once we remove the containers, all the data will be lost. To keep it, we need to configure MongoDB volume and persistent triple store.

4.2.1 MongoDB volume

We use MongoDB to store information about user accounts and access permissions. We can configure a volume so that the data keep on our disk even if we delete MongoDB container.

We can also expose port `27017` so we can access MongoDB from our local computer using a client application like Robo 3T.

Here is the updated docker-compose file:

```
# docker-compose.yml

version: '3'
services:
```

(continues on next page)

(continued from previous page)

```
    fdp:
        image: fairdata/fairdatapoint:1.8.0

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        ports:
            - 80:80
        environment:
            - FDP_HOST=fdp

    mongo:
        image: mongo:4.0.12
        ports:
            - 27017:27017
        volumes:
            - ./mongo/data:/data/db
```

4.2.2 Persistent Repository

FAIR Data Point uses repositories to store the metadata. By default, it uses the in-memory store, which means that the data is lost after the FDP is stopped.

In this example, we will configure Blazegraph as a triple store. See *Triple Stores* for other repository options.

If we don't have it already, we need to create a new file `application.yml`. We will use this file to configure the repository and mount it as a read-only volume to the `fdp` container. This file can be used for other configuration, see *Advanced Configuration* for more details.

```
# application.yml

# ... other configuration

repository:
    type: 5
    blazegraph:
        url: http://blazegraph:8080/blazegraph
```

We now need to update our `docker-compose.yml` file, we add a new volume for the `fdp` and add `blazegraph` service. We can also expose port `8080` for Blazegraph so we can access its user interface.

```
# docker-compose.yml

version: '3'
services:

    fdp:
        image: fairdata/fairdatapoint:1.8.0
        volumes:
            - ./application.yml:/fdp/application.yml:ro

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        ports:
            - 80:80
```

(continues on next page)

(continued from previous page)

```
        environment:
            - FDP_HOST=fdp

    mongo:
        image: mongo:4.0.12
        ports:
            - 27017:27017
        volumes:
            - ./mongo/data:/data/db

    blazegraph:
        image: metaphacts/blazegraph-basic:2.2.0-20160908.003514-6
        ports:
            - 8080:8080
        volumes:
            - ./blazegraph:/blazegraph-data
```

CHAPTER

FIVE

PRODUCTION DEPLOYMENT

If you want to run the FAIR Data Point in production it is recommended to use HTTPS protocol with valid certificates. You can easily configure FDP to run behind a reverse proxy which takes care of the certificates.

In this example, we will configure FDP to run on `https://fdp.example.com`. We will see how to configure the reverse proxy in the same Docker Compose file. However, it is not necessary, and the proxy can be configured elsewhere.

First of all, we need to generate the certificates on the server where we want to run the FDP. You can use Let's Encrypt and create the certificates with certbot. The certificates are generated in a standard location, e.g., `/etc/letsencrypt/live/fdp.example.com` for `fdp.example.com` domain. We will mount the whole `letsencrypt` folder to the reverse proxy container later so that it can use the certificates.

As a reverse proxy, we will use nginx. We need to prepare some configuration, so create a new folder called `nginx` with the following structure and files:

```
nginx/
├ nginx.conf
├ sites-available
│  └ fdp.conf
└ sites-enabled
   └ fdp.conf -> ../sites-available/fdp.conf
```

The file `nginx.conf` is the configuration of the whole nginx, and it includes all the files from `sites-enabled` which contains configuration for individual servers (we can use one nginx, for example, to handle multiple servers on different domains). All available configurations for different servers are in the `sites-available`, but only those linked to `sites-enabled` are used.

Let's see what should be the content of the configuration files.

```
# nginx/nginx.conf

# Main nginx config
user www-data www-data;
worker_processes 5;

events {
    worker_connections 4096;
}

http {
    # Docker DNS resolver
    # We can then use docker container names as hostnames in other configurations
    resolver 127.0.0.11 valid=10s;
```

(continues on next page)

(continued from previous page)

```
    # Include all the configurations files from sites-enabled
    include /etc/nginx/sites-enabled/*.conf;
}
```

Then, we need to configure the FDP server.

```
# nginx/sites-available/fdp.conf

server {
    listen 443 ssl;

    # Generated certificates using certbot, we will mount these in docker-compose.yml
    ssl_certificate /etc/letsencrypt/live/fdp.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/fdp.example.com/privkey.pem;

    server_name fdp.example.com;

    # We pass all the request to the fdp-client container, we can use HTTP in the␣
↪internal network
    # fdp-client_1 is the name of the client container in our configuration, we can␣
↪use it as host
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass_request_headers on;
        proxy_pass http://fdp-client_1;
    }
}

# We redirect all request from HTTP to HTTPS
server {
    listen 80;
    server_name fdp.example.com;
    return 301 https://$host$request_uri;
}
```

Finally, we need to create a soft link from sites-enabled to sites-available for the FDP configuration.

```
$ cd nginx/sites-enabled && ln -s ../sites-available/fdp.conf
```

We have certificates generated and configuration for proxy ready. Now we need to add the proxy to our `docker-compose.yml` file so we can run the whole FDP behind the proxy.

```
# docker-compose.yml

version: '3'
services:
    proxy:
        image: nginx:1.17.3
        ports:
            - 80:80
            - 443:443
        volumes:
            # Mount the nginx folder with the configuration
            - ./nginx:/etc/nginx:ro
            # Mount the letsencrypt certificates
```

(continues on next page)

(continued from previous page)

```
            - /etc/letsencrypt:/etc/letsencrypt:ro

    fdp:
        image: fairdata/fairdatapoint:1.8.0
        volumes:
            - ./application.yml:/fdp/application.yml:ro

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        environment:
            - FDP_HOST=fdp

    mongo:
        image: mongo:4.0.12
        ports:
          - "127.0.0.1:27017:27017"
        volumes:
            - ./mongo/data:/data/db

    blazegraph:
        image: metaphacts/blazegraph-basic:2.2.0-20160908.003514-6
        volumes:
            - ./blazegraph:/blazegraph-data
```

The last thing to do is to update our `application.yml` file. We need to add `clientUrl` so that FDP knows the actual URL even if hidden behind the reverse proxy. It's a good practice to set up a persistent URL for the metadata too. We recommend using `https://purl.org`. If you don't specify `persistentUrl`, the `clientUrl` will be used instead. And we also need to set a random JWT token for security.

```
# application.yml

instance:
    clientUrl: https://fdp.example.com
    persistentUrl: https://purl.org/fairdatapoint/example

security:
    jwt:
        token:
            secret-key: <random 128 characters string>

# repository settings (can be changed to different repository)
repository:
    type: 5
    blazegraph:
        url: http://blazegraph:8080/blazegraph
```

At this point, we should be able to run all the containers using `docker-compose up -d` and after everything starts, we can access the FAIR Data Point at https://fdp.example.com. Of course, the domain you want to access the FDP on must be configured to the server where it runs.

Danger: Don't forget to change the default user accounts as soon as your FAIR Data Point becomes publicly available.

Danger: Do not expose mongo port unless you secured the database with username and password.

Warning: In order to improve findability of itself and its content, the FAIR Data Point has a built-in feature that registers its URL into our server and pings it once a week. This feature facilitates the indexing of the metadata of each registered and active FAIR Data Point. If you do not want your FAIR Data Point to be included in this registry, add these lines to your application configuration:

```
# application.yml

ping:
    enabled: false
```

CHAPTER

SIX

ADVANCED CONFIGURATION

6.1 Triple Stores

FDP uses InMemory triple store by default. In previous examples, there is Blazegraph used. However, you can choose from 3 additional options.

List of possible triple stores:

1. In-Memory Store
2. Native Store
3. Allegro Graph Repository
4. GraphDB Repository
5. Blazegraph Repository

6.1.1 1. In-Memory Store

There is no need to configure additional properties to run FDP with In-Memory Store because it's the default option. If you want to explicitly type in configuration provided in `application.yml`, add following lines there:

```
# application.yml

repository:
    type: 1
```

6.1.2 2. Native Store

With this option, FDP will simply save the data to the file system. If you want to use the Native Store, make sure that you have these lines in your `application.yml` file:

```
# application.yml

repository:
    type: 2
    native:
        dir: /tmp/fdp-store
```

where `/tmp/fdp-store` is a path to a location where you want to keep your data stored.

6.1.3 3. Allegro Graph

For running Allegro Graph, you need to first set up your Allegro Graph instance. For configuring the connection from FDP, add these lines to your `application.yml` file:

```
# application.yml

repository:
    type: 3
    agraph:
        url: http://agraph:10035/repositories/fdp
        username: user
        password: password
```

`URL`, `username` and `password` should be configured according to your actual Allegro Graph setup.

6.1.4 4. GraphDB

For running GraphDB, you need to first set up your GraphDB instance and **create the repository**. For configuring the connection from FDP, add these lines to your `application.yml` file:

```
# application.yml

repository:
    type: 4
    graphDb:
        url: http://graphdb:7200
        repository: <repository-name>
```

`URL` and `repository` should be configured according to your actual GraphDB setup.

6.1.5 5. Blazegraph

For running Blazegraph, you need to first set up your Blazegraph instance. For configuring the connection from FDP, add these lines to your `application.yml` file:

```
# application.yml

repository:
    type: 5
    blazegraph:
        url: http://blazegraph:8080/blazegraph
        repository:
```

`URL` and `repository` should be configured according to your actual Blazegraph setup. Repository should be set only if you don't use the default one.

6.2 Mongo DB

We store users, permissions, etc. in the MongoDB database. The default connection string is `mongodb://mongo:27017/fdp`. If you want to modify it, add these lines to your `application.yml` file:

```
# application.yml

spring:
    data:
        mongodb:
            uri: mongodb://mongo:27017/fdp
```

The `uri` should be adjusted by your actual MongoDB setup.

6.3 Default attached metadata

There are several default values that are attached to each created metadata. If you want to modify it, add the lines below to your `application.yml` file. The default values are listed below, too:

```
# application.yml

metadataProperties:
    language: http://id.loc.gov/vocabulary/iso639-1/en
    license: http://rdflicense.appspot.com/rdflicense/cc-by-nc-nd3.0
    accessRightsDescription: This resource has no access restriction

metadataMetrics:
    https://purl.org/fair-metrics/FM_F1A: https://www.ietf.org/rfc/rfc3986.txt
    https://purl.org/fair-metrics/FM_A1.1: https://www.wikidata.org/wiki/Q8777
```

6.4 Customizations

You can customize the look and feel of FDP Client using SCSS. There are three files you can mount to `/src/scss/custom`. If there are any changes in these files, the styles will be regenerated when FDP Client starts.

6.4.1 Customization files

_variables.scss

A lot of values related to styles are defined as variables. The easiest way to customize the FDP Client is to define new values for these variables. To do so, you create a file called `_variables.scss` where you define the values that you want to change.

Here is an example of changing the primary color.

```
// _variables.scss

$color-primary: #087d63;
```

Have a look in src/scss/_variables.scss to see all the variables you can change.

_extra.scss

This file is loaded before all other styles. You can use it, for example, to define new styles or import fonts.

_overrides.scss

This file is loaded after all other styles. You can use it to override existing styles.

6.4.2 Example of setting a custom logo

To change the logo, you need to do three steps:

1. Create `_variables.scss` with correct logo file name and dimensions
2. Mount the new logo to the assets folder
3. Mount `_variables.scss` to SCSS custom folder

```
// _variables.scss

$header-logo-url: '/assets/my-logo.png';  // new logo file
$header-logo-width: 80px;  // width of the new logo
$header-logo-height: 40px;  // height of the new logo
```

```
# docker-compose.yml

version: '3'
services:
    fdp:
        # ... FDP configuration

    fdp-client:
        # ... FDP Client configuration
        volumes:
          # Mount new logo file to assets in the container
          - ./my-logo.png:/usr/share/nginx/html/assets/my-logo.png:ro

          # Mount _variables.scss so that styles are regenerated
          - ./_variables.scss:/src/scss/custom/_variables.scss:ro
```

6.5 Running FDP on a nested route

Sometimes, you might want to run FDP alongside other applications on the same domain. Here is an example of running FDP on `https://example.com/fairdatapoint`. If you run FDP in this configuration, you have to set `PUBLIC\_PATH` ENV variable, in this example to `/fairdatapoint`. Also, don't forget to set correct client URL in the application config.

```
# docker-compose.yml

version: '3'
services:
    fdp:
        image: fairdata/fairdatapoint:1.8.0
```

(continues on next page)

(continued from previous page)

```
        volumes:
            - ./application.yml:/fdp/application.yml:ro
            # ... other volumes

    fdp-client:
        image: fairdata/fairdatapoint-client:1.8.0
        ports:
            - 80:80
        environment:
            - FDP_HOST=fdp
            - PUBLIC_PATH=/fairdatapoint
```

```
# application.yml

instance:
    clientUrl: https://example.com/fairdatapoint
```

```
# Snippet for nginx configuration

server {
    # Configruation for the server, certificates, etc.

    # Define the location FDP runs on
    location ~ /fairdatapoint(/.*)?$ {
        rewrite /fairdatapoint(/.*) $1 break;
        rewrite /fairdatapoint / break;
        proxy_pass http://<client_host>;
    }
}
```

When running on nested route, don't forget to change paths to all custom assets referenced in SCSS files.

CHAPTER

SEVEN

USAGE

7.1 Resource definitions

The **FAIR Data Point** reference implementations introduces the concept of **Resource Definitions**. A resource definition captures housekeeping data about a metadata resource.

Resource definitions can be accessed from the reference implementation user interface, in the dashboard of an admin user. Here the resource definitions can be managed.

7.1.1 Managing resource definitions

Resource definitions can be *created*, *modified*, or *deleted*.

Creating resource definitions

When creating a new resource definition, the user interface presents you a form where a resource can be defined through a number of properties.

Name defines the human readable name for a resource definition. This name is used in the admin dashboard to identify the definitions, and does not affect a definition on the functional level. For example, for the default `dcat:Dataset` resource, the human readable name would be `"Dataset"`.

URL Prefix defines the URL path for a resource. This context path should be a unique identifier, unique in the scope of the other resource defined within the FAIR Data Point instance. For example, for the `dcat:Dataset` resource the prefix is `dataset`.

Target Class URIs links the *shape definitions* to a resource definition. In the current implementation, each shape that should be applied to the resource must be listed here. The expected URI value must match the `sh:targetClass` value of the shape definition. A common example is to list the `dcat:Resource` shape target class along with the specific subclass for the resource, like `dcat:Dataset`.

Children defines child resources, if any. This applies when the resource acts as a parent resource for other types of resources. Children are defined by the following properties to provide directives for both the server and the client side.

- *Child Resource* links to the child's *resource definition*. The child resource must be defined beforehand.
- *Child Relation URI* defines the predicate IRI that links the parent to the child on the metadata instance level. A common example is the link from a `dcat:Dataset` to a `dcat:Distribution`; these resources are linked by the `dcat:distribution` predicate.
- *Child List View Title* defines a literal value to be displayed as a section header for the child resources in the user interface.

- *Child List View Tags URI* defines the predicate IRI for values that are displayed in the user interface whenever the child resources are listed. A common example is `dcat:theme` for `dcat:Dataset` resources.
- *Child List View Metadata* defines predicate IRIs for values that are listed in the child resource summary.

External links defines predicate IRIs and literal values to be displayed in the user interface for primary interaction with a resource. A common example is for the `dcat:Distribution` resource, `dcat:accessURL` is mapped to an `Access URL` literal, and displayed prominently in the user interface.

Modifying resource definitions

When modifying a resource definition, not all properties are writable. Some properties are write-protected to ensure the internal consistency of the system.

The **URL Prefix** and the **Target Class URIs** are not writable.

The other properties, like child resources and external links are writable, and can be expanded or modified after the initial creation of the resource definition.

Deleting resource definitions

Deleting a resource definition should be used with caution. Existing metadata instances are no longer accessible after the resource definition is deleted. This includes child resources, if those are not linked to other resources.

7.2 Shapes

The **FAIR Data Point** reference implementation uses SHACL to add validation constraints to the metadata model.

7.2.1 Creating a new shape

A typical resource shape contains the following key elements:

- `sh:targetClass` to indicate the type of resource the shape applies to
- `sh:property` for each resource property
  - `sh:path` defines the predicate IRI
  - `sh:nodeKind`/`sh:datatype` defines the object type
  - `sh:minCount`/`sh:maxCount` defines the property cardinality

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix dash: <http://datashapes.org/dash#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix ex: <http://example.com/> .

:MyResourceShape a sh:NodeShape ;
    sh:targetClass ex:MyResourceType ;
    sh:property [
        sh:path ex:value ;
        sh:nodeKind sh:Literal ;
        sh:minCount 1 ;
        sh:maxCount 2 ;
    ] .
```

7.2.2 User interface directives

The DASH vocabulary introduces extensions to the core SHACL model. One of the extensions is focused on providing user interface hints for shape properties. Introducing or removing a `dash:viewer` or `dash:editor` property to a `sh:PropertyShape` instance influences how the user interface displays the property value.

```
sh:property [
    sh:path ex:value ;
    sh:nodeKind sh:Literal ;
    dash:viewer dash:LiteralViewer ;
    dash:editor dash:TextFieldEditor ;
]
```

By adding a `dash:viewer` statement, the user interface is instructed to show the property value when the resource metadata is displayed. Removing a `dash:viewer` statement will instruct the user interface will not render the property value at all. The value will still be present in the metadata model. The supported set of viewers:

- `sh:LabelViewer`
- `sh:URIViewer`

By adding a `dash:editor` statement, the editor form in the user interface will show an edit field for the property. Removing a `dash:editor` statement will prevent the property from being edited. This could be intended behaviour for properties that are generated server side. The supported set of editors:

- `sh:TextFieldEditor`
- `sh:TextAreaEditor`
- `sh:URIEditor`
- `sh:DatePickerEditor`

7.2.3 Extending an existing shape

Extending an existing shape can be achieved by targeting the same `sh:targetClass`. For example, to extend the existing `dcat:Dataset` shape, an extension shape could look like the following:

```
:MyExtension a sh:NodeShape ;
    sh:targetClass dcat:Dataset ;
    sh:property [
        sh:path <http://example.com/vocab#myProperty> ;
        sh:nodeKind sh:Literal ;
        sh:minCount 1 ;
    ] .
```

7.2.4 Limitations

- The current implementation does not provide proper support for overriding properties when an existing resource is extended
- The set of supported `dash:viewer` and `dash:editor` types does not cover the full range as specified in the DASH specs.

CHAPTER

EIGHT

API USAGE

The **FAIR Data Point** exposes API endpoints that allow consumers to interact with the metadata. Some of the endpoints are available for all users, while others require an API token for authorization.

8.1 Obtaining an API token

In order to obtain an API token, you invoke the `/tokens` endpoint with your user credentials.

```
curl -X POST -H "Accept: application/json" \
    -H "Content-Type: application/json" \
    -d '{ "email": "user@example.com", "password": "secret" }' \
    https://fdp.example.com/tokens
```

A successful call will return a `JSON` object with a token.

```
{ "token": "efIobn394nvJJFJ30..." }
```

8.1.1 Issueing a request with the authorization token

Subsequent requests should use this token in the *Authorization* header. The authorization type is a Bearer token.

```
curl -H "Authorization: Bearer efIobn394nvJJFJ30..." \
    -H "Accept: application/json" https://fdp.example.com/users
```

8.2 Interacting with metadata

The metadata layers as defined by the *Resource definitions* are exposed through their respective endpoints. The general approach is that each layer, defined by its `prefix`, supports a number of read and write HTTP methods.

Method	URL pattern	Functionality
`GET`	`/${prefix}/${uuid}`	*Retrieving metadata*
`POST`	`/${prefix}`	*Creating metadata*
`PUT`	`/${prefix}/${uuid}`	*Update metadata*

8.2.1 Retrieving metadata

Retrieving metadata is open for GET requests without authorization. In the following example, we retrieve a `Dataset` resource by issuing a `GET` request to the `/dataset` prefix followed by its identifier (a UUID).

```
curl -H "Accept: text/turtle" https://fdp.example.com/dataset/58d7fbde-6c16-483e-b152-
↪0f3ced131ca9
```

8.2.2 Creating metadata

New metadata can be created by `POST`-ing the content to the appropriate endpoint. First we will create a file called `metadata.ttl` to store our new metadata.

```
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<> a dcat:Dataset ;
    dct:title "test" ;
    dct:hasVersion "1.0" ;
    dct:publisher [ a foaf:Agent ; foaf:name "Example User" ] ;
    dcat:theme <http://www.wikidata.org/entity/Q14944328> ;
    dct:isPartOf <https://fdp.example.com/catalog/5f4a32c5-1f26-4657-9240-
↪fc7ede7f1ce5> .
```

This metadata can be created by the following `POST` request.

```
curl -H "Authorization: Bearer efIobn394nvJJFJ30..." \
    -H "Content-Type: text/turtle" \
    -d @metadata.ttl https://fdp.example.com/dataset
```

When created, the metadata is initially in a `DRAFT` state. To publish the metadata using the API you can issue the following `PUT` request to transistion the metadata from the `DRAFT` state to the `PUBLISHED` state.

```
curl -X PUT -H "Authorization: Bearer efIobn394nvJJFJ30..." \
    -H "Accept: application/json" \
    -H "Content-Type: application/json" \
    -d '{ "current": "PUBLISHED" }' \
    https://fdp.example.com/dataset/79508287-a2a7-4ae2-95b3-3f595e3088cc/meta/state
```

8.2.3 Update metadata

Existing metadata can be updated by issuing a `PUT` request with the request body being the updated metadata.

```
curl -X PUT -H "Authorization: Bearer efIobn394nvJJFJ30..." \
    -H "Content-Type: text/turtle" \
    -d @metadata.ttl https://fdp.example.com/dataset/79508287-a2a7-4ae2-95b3-
↪3f595e3088cc
```

8.3 API endpoint listing

The available APIs are documented using OpenAPI. In the `/swagger-ui.html` endpoint the APIs are visualized through Swagger UI.

CHAPTER

NINE

USAGE

Here you can read how to use the **metadata** extension for OpenRefine to store FAIR data and create metadata in FAIR Data Point.

9.1 About metadata extension

The **metadata** extension for OpenRefine promotes FAIRness of the data by its integration with FAIR Data Point. With the extension you can easily FAIRify your data that you work on in directly in OpenRefine in two steps:

1. *Store FAIR data* in configured storage.
2. *Create metadata in FAIR Data Point* in selected FAIR Data Point.

It replaces the legacy project called FAIRifier.

9.2 Features

The extension provides the features only through *FAIR Metadata* extension menu located in top right corner above data table (typically next to *Wikidata* and others).

9.2.1 Store FAIR data

1. Open the dialog for storing the data by clicking *FAIR Metadata* > *Store data to FAIR storage*
2. Select the desired storage (see *Storages*)
3. Select the desired format (the selection changes based on storage)
4. Press *Preview (download)* to download the file to verify the contents
5. Press *Store* to store the data in the storage
6. You will see the URL to the file which you can easy copy to clipboard by clicking a button

9.2.2 Create metadata in FAIR Data Point

1. Open the dialog for creating the metadata by clicking *FAIRMetadata* > *Create metadata in FAIR Data Point*
2. Select pre-configured FAIR Data Point connection or select *Custom FDP connection* and fill information (if allowed, see *Settings*)
3. Press *Connect* to connect to the selected FAIR Data Point
4. Select a catalog from available or create a new one
    - For a new one, fill in the metadata form (see also the optional fields) and press *Create catalog*
5. Select a dataset from available or create a new one
    - For a new one, fill in the metadata form (see also the optional fields) and press *Create dataset*
5. Create a new distribution
6. Fill in the metadata form (se also the optional fiels)
    - For the download URL you can easily access *Store FAIR data* feature and field will be filled after storing the data
7. Check your new distribution (and/or other layers) listed

CHAPTER

TEN

SETUP

This part describes how to set up your own OpenRefine with the **metadata** extension and how to configure it according to your needs.

10.1 Installation

There are two ways of using our **metadata** extension for OpenRefine. You can have installed OpenRefine and add extension to it or use Docker with our prepared image.

10.1.1 Installed OpenRefine

This option requires you to have installed compatible version of OpenRefine, please check *Compatibility*. In case you need to install OpenRefine first, visit their documentation.

- Get the desired version of the **metadata** extension from our GitHub releases page by downloading tgz or zip archive, e.g., `metadata-X.Y.Z-OpenRefine-X.Y.zip`.
- Extract the archive to `extensions` folder of your OpenRefine (see OpenRefine documentation).

```
unzip metadata-X.Y.Z-OpenRefine-X.Y.zip path/to/openrefine-X.Y/webapp/extensions
```

10.1.2 With Docker

If you want to use Docker, we provide a Docker image fairdata/openrefine-metadata-extension that combines the extension with OpenRefine of supported version. It is of course possible to use volume for the `data` directory (eventually `data/extensions` to include other extensions). All you need to have is Docker running and then:

```
docker run -p 3333:3333 -v /home/me/openrefine-data:/data:z fairdata/openrefine-
↪metadata-extension
```

This will run the OpenRefine with **metadata** extension on port 3333 that will be exposed and mounts your folder `/home/me/openrefine-data` as OpenRefine data folder. You should be able to open OpenRefine in browser on `localhost:3333`. If there are some other extensions in `/home/me/openrefine-data/extensions`, those should be loaded as well. For more information, see OpenRefine documentation.

For configuration files you need to mount `/webapp/extensions/metadata/module/config`, see *Configuration* for more details.

10.2 Configuration

Configuration files of the **metadata** extension use the YAML (YAML Ain't Markup Language) format and are stored in `extensions/metadata/module/config` directory of the used OpenRefine installment. The configuration files are loaded when OpenRefine is started. Therefore, you are required to restart OpenRefine before changes in configuration files take effect. We provide examples of the configuration files that you can (re)use.

10.2.1 Settings

Settings configuration file serves for generic configuration options that adjust behaviour of the extension. The structure of the file is following:

- `allowCustomFDP` (boolean) = should be user allowed to enter custom FAIR Data Point URI (Uniform Resource Identifier), username, and password (or use only the pre-configured)
- `metadata` (map) = key-value specification of instance-wide pre-defined metadata, e.g., set `license` to `http://purl.org/NET/rdflicense/cc-by3.0` and that URI will be pre-set in all metadata forms in field `license` (but can be overwritten by the user)
- `fdpConnections` (list) = list of pre-configured FAIR Data Point connections that users can use, each is object with attributes:
  - `name` (string) = custom name identifying the connection
  - `baseURI` (string) = base URI of FAIR Data Point
  - `email` (string) = email address identifying user of FAIR Data Point
  - `password` (string) = password for authenticating the user of FAIR Data Point
  - `preselected` (boolean, optional) = flag if should be pre-selected in the form (in case that more connections have this set to true, only first one is applied)
  - `metadata` (map, optional) = similar to instance-wide but only for specific connection

For more information and further configuration options, see examples.

10.2.2 Storages

Storages configuration file holds details about storages that are possible to use for *Store FAIR data* feature. In the file, list of storage object is expected where each of them has:

- `name` (string) = custom name identifying the storage
- `type` (string) = one of the allowed types (others are ignored): `ftp`, `virtuso`, `tripleStoreHTTP`
- `details` (object) = configuration related to specific type of storage (see examples)

For FTP (File Transfer Protocol) and Virtuoso, `directory` should containt absolute path where files should be stored. In case of triple stores, repository name is used to specify the target location.

10.3 Compatibility

Check in-app “About” dialog for compatibility information.

CHAPTER

ELEVEN

CONTRIBUTING

11.1 Development

Our projects are open source and you can contribute via GitHub (fork and pull request):

- https://github.com/FAIRDataTeam/FAIRDataPoint
- https://github.com/FAIRDataTeam/FAIRDataPoint-client
- https://github.com/FAIRDataTeam/OpenRefine-metadata-extension

CHAPTER

TWELVE

CHANGELOG

12.1 Overview

Here we summarize the key features and changes for each FAIR Data Point release. For details including bugfixes and minor changes, see *Detailed changelog*.

12.1.1 1.8.0

- Added Admin UI to FDP Index with possibility to trigger metadata retrieval, change settings, or delete entry
- Several bug fixes and dependencies updated (including Java 15)

12.1.2 1.7.0

- Including FDP Index functionality into FAIR Data Point with harvesting metadata
- Metadata search including RDF types
- Possibility to change profile and password for current user

12.1.3 1.6.0

- API keys for making integrations with FDP easier
- State “draft” for created metadata

12.1.4 1.5.0

- Support for editable resource definitions
- Possibility to specify custom storage in OpenRefine using frontend

12.1.5 1.4.0

- Ping service for *call home* functionality
- Suggesting prefixes for namespaces

12.1.6 1.3.0

- Introduced DASH and dynamic SHACL shapes
- Audit log in OpenRefine extension to keep track of actions performed

12.1.7 1.2.0

- Option to customize metamodel (metadata layers)
- Possibility to delete and create metadata entities

12.1.8 1.1.0

- New monitoring and configuration for client application
- Several further improvements in terms of technical debt
- Enhanced connecting to FDP from OpenRefine extension and update to OpenRefine 3.3

12.1.9 1.0.0

- User management, enhanced security, and ACL
- Huge refactoring and upgrades of previously accumulated features and technical debt
- Separate project for FAIR Data Point Client (frontend application using FDP API)
- New OpenRefine Metadata Extension as a replacement for the deprecated FAIRifier

12.2 Detailed changelog

Each of components developed has its own Changelog based on Keep a Changelog, and our projects adhere to Semantic Versioning. It is recommended to use matching versions of all components.

- FAIR Data Point Changelog
- FAIR Data Point Client Changelog
- OpenRefine Metadata Extensions Changelog