ehrbase Documentation

Birger Haarbrandt

Jul 03, 2020

Contents:

For now, there is only the Release Notes. We will add detailed documentation as soon as possible.

CHAPTER 1

Release Notes

The following pages show the release notes

1.1 Release Notes EHRbase 0.12.0 (alpha)

This release of EHRbase (v0.12.0, March 31 2020) adds basic authentication (see details below) and allows to overwrite templates. we consider EHRbase to be still in alpha status.

The following changes are included in this version:

1.1.1 Added

- Basic Authentication as opt-in (see: https://github.com/ehrbase/ehrbase/pull/200)
- Allow Templates can now be overwritten via spring configuration (see: https://github.com/ehrbase/ehrbase/pull/194)
- Prototypical support of FHIR Terminology Services within AQL queries (see: https://github.com/ehrbase/ehrbase/pull/162)

1.1.2 Fixed

- Fixes response code on /ehr PUT with invalid ID (see: https://github.com/ehrbase/project_management/issues/163)
- Fixes STATUS w/ empty subject bug (see: https://github.com/ehrbase/ehrbase/pull/196)
- Fixes storage of party self inside compositions (see: https://github.com/ehrbase/ehrbase/pull/195)
- Added support of AQL query in the form of c/composer (see: https://github.com/ehrbase/ehrbase/pull/184)
- Java error with UTF-8 encoding resolved (see: https://github.com/ehrbase/ehrbase/pull/173)

- AQL refactoring and fixes to support correct canonical json representation (see: https://github.com/ehrbase/ehrbase/pull/201)
- fix terminal value test for non DataValue 'value' attribute (see: https://github.com/ehrbase/ehrbase/pull/189)

1.1.3 General Features

- openEHR Reference Model Version 1.0.4
- Serialisation of Reference Model Objects in Canonical JSON and XML
- Archetype Definition Language 1.4
- Data Validation against Operational Templates
- openEHR REST API Endpoints (see below for details)

openEHR REST API

Based on the official openEHR REST API the following endpoints are implemented:

- EHR (CREATE EHR, CREATE EHR with Id)
- EHR_STATUS
- COMPOSITION (Create, Update, Delete, Get Composition by Version Id, Get composition at time)
- CONTRIBUTION (Create, Get of compositions. Other versioned object like EHR_STATUS coming soonly)
- DIRECTORY (Create, Update, Delete, Get folder in directory version, get folder in directory version at time)
- QUERY (Execute ad-hoc (non-stored) AQL Query, Execute stored query, parameters))
- STORED_QUERY (List Stored Queries, Store a query, Get stored query, delete, parameters)
- ADL 1.4 TEMPLATE (Upload a Template, List Templates, Get Template)

Note: The Swagger UI is generally WIP and currently does not distinguish between implemented endpoints and stubs! This means that you will see some endpoints that you cannot use)

Note: The data format for contributions sent through the REST API is not yet defined in the openEHR. Please refer to the examples. Also note that the format might be subject of change.

Conformance Tests

EHRbase ships with a set of tests verifying the conformance with the openEHR REST API. For now the tests include the following endpoints:

- EHR
- EHR_STATUS
- COMPOSITION
- CONTRIBUTION
- ADL 1.4 TEMPLATE

- DIRECTORY
- QUERY

1.1.4 What (basic) features you might miss

- VERSIONED_OBJECT Endpoints are not implemented
- EHR functions like is_modifyable and is_queryable are not yet supported

1.1.5 Known Issues

As EHRbase is still in alpha status, there are plenty of known issues. If you try things out, please be aware that the following issues are known and documented:

Archetype Query Language

- ehr e projection not supported
- Not supported variables in archetype_id predicates

```
select e/ehr_id/value, e/time_created/value, e/system_id/value from EHR
e CONTAINS COMPOSITION c [$archetype_id]
```

- TIMEWINDOW keyword is not supported

```
SELECT e/ehr_id/value FROM EHR e TIMEWINDOW PT12H/2019-10-24
```

1.2 Release Notes EHRbase 0.11.0 (alpha)

This release of EHRbase (2020-03-03) provides several improvements, especially regarding stability and bug fixes. However, we consider EHRbase to be still in alpha status.

The following changes are included in this version:

1.2.1 Added

- Docker and docker-compose support for both application and database
- Get folder with version_at_time parameter
- Get Folder with path parameter

1.2.2 Changed

- FasterXML Jackson version raised to 2.10.2
- Java version raised from 8 to 11
- Jooq version raised to 3.12.3
- Spring Boot raised to version 2

1.2.3 Fixed

- Response code when composition is logically deleted (see: https://github.com/ehrbase/ehrbase/pull/144)
- Response and *PREFER* header handling of */ehr* endpoints (see: https://github.com/ehrbase/ehrbase/pull/165)
- Deserialization of EhrStatus attributes is_modifiable and is_queryable are defaulting to *true* now (see: https://github.com/ehrbase/ehrbase/pull/158)
- Updating of composition with invalid template (e.g. completely different template than the previous version) (see: https://github.com/ehrbase/ehrbase/pull/166)
- Folder names are checked for duplicates (see: https://github.com/ehrbase/ehrbase/pull/168)
- AQL parser threw an unspecific exception when an alias was used in a WHERE clause (https://github.com/ehrbase/ehrbase/pull/149)
- Improved exception handling in composition validation (see: https://github.com/ehrbase/ehrbase/pull/147)
- Improved Reference Model validation (see: https://github.com/ehrbase/ehrbase/pull/147)
- Error when reading a composition that has a provider name set(see: https://github.com/ehrbase/ehrbase/pull/143)
- Allow content to be null inside a composition (see: https://github.com/ehrbase/ehrbase/pull/129)
- Fixed deletion of compositions through a contribution (see: https://github.com/ehrbase/ehrbase/pull/128)
- Start time of a composition was not properly updated (see: https://github.com/ehrbase/ehrbase/pull/137)
- Fixed validation of null values on participations (see: https://github.com/ehrbase/ehrbase/pull/132)
- Order by in AQL did not work properly (see: https://github.com/ehrbase/ehrbase/pull/112)
- Order of variables in AQL result was not preserved (see: https://github.com/ehrbase/ehrbase/pull/103)
- Validation of compositions for unsupported language(see: https://github.com/ehrbase/ehrbase/pull/107)
- Duplicated ehr attributes in query due to cartesian product (see: https://github.com/ehrbase/ehrbase/pull/106)
- Retrieve of EHR_STATUS gave Null Pointer Exception for non-existing EHRs (see: https://github.com/ehrbase/ehrbase/pull/136)
- Correct resolution of ehr/system_id in AQL (see: https://github.com/ehrbase/ehrbase/pull/102)
- Detection of duplicate aliases in aql select (see: https://github.com/ehrbase/ehrbase/pull/98)

1.2.4 General Features

- openEHR Reference Model Version 1.0.4
- Serialisation of Reference Model Objects in Canonical JSON and XML
- Archetype Definition Language 1.4
- Data Validation against Operational Templates
- openEHR REST API Endpoints (see below for details)

openEHR REST API

Based on the official openEHR REST API the following endpoints are implemented:

- EHR (CREATE EHR, CREATE EHR with Id)

- EHR_STATUS
- COMPOSITION (Create, Update, Delete, Get Composition by Version Id, Get composition at time)
- CONTRIBUTION (Create, Get of compositions. Other versioned object like EHR_STATUS coming soonly)
- DIRECTORY (Create, Update, Delete, Get folder in directory version, get folder in directory version at time)
- QUERY (Execute ad-hoc (non-stored) AQL Query, Execute stored query, parameters))
- STORED_QUERY (List Stored Queries, Store a query, Get stored query, delete, parameters)
- ADL 1.4 TEMPLATE (Upload a Template, List Templates, Get Template)

Note: The Swagger UI is generally WIP and currently does not distinguish between implemented endpoints and stubs! This means that you will see some endpoints that you cannot use)

Note: The data format for contributions sent through the REST API is not yet defined in the openEHR. Please refer to the examples. Also note that the format might be subject of change.

Conformance Tests

EHRbase ships with a set of tests verifying the conformance with the openEHR REST API. For now the tests include the following endpoints:

- EHR
- EHR_STATUS
- COMPOSITION
- CONTRIBUTION
- ADL 1.4 TEMPLATE
- DIRECTORY
- QUERY

1.2.5 What (basic) features you might miss

- VERSIONED_OBJECT Endpoints are not implemented
- Authentication is not implemented (planned to be implemented using Spring Security)
- Connection to external terminology service (like FHIR TS) is not yet supported
- EHR functions like is_modifyable and is_queryable are not yet supported

1.2.6 Known Issues

As EHRbase is still in alpha status, there are plenty of known issues. If you try things out, please be aware that the following issues are known and documented:

Archetype Query Language

- ehr e projection not supported
- Not supported variables in archetype_id predicates

```
select e/ehr_id/value, e/time_created/value, e/system_id/value from EHR
e CONTAINS COMPOSITION c [$archetype_id]
```

- TIMEWINDOW keyword is not supported

```
SELECT e/ehr_id/value FROM EHR e TIMEWINDOW PT12H/2019-10-24
```

1.3 Release Notes EHRbase 0.10.0 (alpha)

This is the alpha release of EHRbase (2020-01-10). This means that the basic set of functions has been implemented. Please be aware that the current state can be used to develop openEHR applications but should not be used for real patient data in production.

The following changes are included in this version:

- openEHR REST API DIRECTORY Endpoints
- openEHR REST API EHR_STATUS Endpoints (including other_details)
- Spring Transactions: EHRbase now ensures complete rollback if part of a transaction fails.
- Improved Template storage: openEHR Templates are stored inside the postgres database instead of the file system (including handling of duplicates)
- AQL queries with partial paths return data in canonical json format (including full compositions)
- Multimedia data can be correctly stored and retrieved
- Spring configuration allows setting the System ID
- Validation of openEHR Terminology (openEHR terminology codes are tested against an internal terminology service)
- Order of columns in AQL result sets are now reliably preserved (https://github.com/ehrbase/ehrbase/issues/37)
- Some projection issues for EHR attributes have been resolved in AQL
- Fixed error regarding DISTINCT operator in AQL (https://github.com/ehrbase/ehrbase/issues/50)
- Fixed null pointer exceptions that could occur in persistent compositions
- Lots of new conformance tests for QUERY Endpoints

1.3.1 General Features

- openEHR Reference Model Version 1.0.4
- Serialisation of Reference Model Objects in Canonical JSON and XML
- Archetype Definition Language 1.4
- Data Validation against Operational Templates
- openEHR REST API Endpoints (see below for details)

openEHR REST API

Based on the official openEHR REST API the following endpoints are implemented:

- EHR (CREATE EHR, CREATE EHR with Id)
- EHR_STATUS
- COMPOSITION (Create, Update, Delete, Get Composition by Version Id, Get composition at time)
- CONTRIBUTION (Create, Get of compositions. Other versioned object like EHR_STATUS coming soonly)
- DIRECTORY
- QUERY (Execute ad-hoc (non-stored) AQL Query, Execute stored query, parameters))
- STORED_QUERY (List Stored Queries, Store a query, Get stored query, delete, parameters)
- ADL 1.4 TEMPLATE (Upload a Template, List Templates, Get Template)

Note: The Swagger UI is generally WIP and currently does not distinguish between implemented endpoints and stubs! This means that you will see some endpoints that you cannot use)

Note: The data format for contributions sent through the REST API is not yet defined in the openEHR. Please refer to the examples. Also note that the format might be subject of change.

Conformance Tests

EHRbase ships with a set of tests verifying the conformance with the openEHR REST API. For now the tests include the following endpoints:

- EHR
- EHR_STATUS
- COMPOSITION
- CONTRIBUTION
- ADL 1.4 TEMPLATE
- DIRECTORY
- QUERY

1.3.2 What (basic) features you might miss

- VERSIONED_OBJECT Endpoints are not implemented
- Authentication is not implemented (planned to be implemented using Spring Security)
- Connection to external terminology service (like FHIR TS) is not yet supported
- EHR functions like is_modifyable and is_queryable are not yet supported

1.3.3 Known Issues

As EHRbase is still in alpha status, there are plenty of known issues. If you try things out, please be aware that the following issues are known and documented:

Archetype Query Language

- ehr/uid projection not supported (EHRBASE supports ehr/uid/value but not ehr/uid)

```
SELECT e/uid, e/time_created, e/system_id FROM EHR e
```

- Not supported variables in archetype_id predicates

```
select e/ehr_id/value, e/time_created/value, e/system_id/value from EHR
e CONTAINS COMPOSITION c [$archetype_id]
```

- TIMEWINDOW keyword is not supported

```
SELECT e/ehr_id/value FROM EHR e TIMEWINDOW PT12H/2019-10-24
```

1.4 Release Notes EHRbase 0.9.0 (pre-alpha)

As the initial open source version of EHRbase, this release note is a bit more comprehensive. Please be aware that this is a pre-alpha version. We will soonly release the alpha version containing EHR_STATUS and DIRECTORY Endpoints of the official openEHR REST API. Please be aware that the current state can be used to develop openEHR applications but should not be used for real patient data in production.

Here is an overview of the features implemented:

1.4.1 Features

- openEHR Reference Model Version 1.0.4
- Serialisation of Reference Model Objects in Canonical JSON and XML
- Archetype Definition Language 1.4
- Data Validation against Operational Templates

openEHR REST API

Based on the official openEHR REST API the following endpoints are implemented:

- EHR (CREATE EHR, CREATE EHR with Id)
- COMPOSITION (Create, Update, Delete, Get Composition by Version Id, Get composition at time)
- CONTRIBUTION (Create, Get of compositions. Other versioned object like EHR_STATUS coming soonly)
- QUERY (Execute ad-hoc (non-stored) AQL Query, Execute stored query))
- STORED_QUERY (List Stored Queries, Store a query, Get stored query, delete)
- ADL 1.4 TEMPLATE (Upload a Template, List Templates, Get Template)

Note: The Swagger UI is generally WIP and currently does not distinguish between implemented endpoints and stubs! This means that you will see some endpoints that you cannot use)

Note: The data format for contributions sent through the REST API is not yet defined in the openEHR. Please refer to the examples. Also note that the format might be subject of change.

Conformance Tests

EHRbase ships with a set of tests verifying the conformance with the openEHR REST API. For now the tests include the following endpoints:

- EHR
- COMPOSITION
- CONTRIBUTION
- ADL 1.4 TEMPLATE
- DIRECTORY

Java Client Library

The client library has the following features:

- Create EHR
- Upload/Sync Templates
- Send Composition
- Creation of Java (DTO) classes from Operational Templates (OPTs)

1.4.2 What you might miss

- DIRECTORY Endpoints will be added soonly
- EHR_STATUS Endpoints will be added soonly
- VERSIONED_STATUS Endpoints are not implemented
- Authentication is not implemented (planned to be implemented using Spring Security)
- Connection to external terminology service (like FHIR TS) is not yet supported
- EHR functions like is_modifyable and is_queryable are not yet supported
- Database transactions and rollback (planned to implemented using Spring Transaction Management)

1.4.3 Known Issues

As EHRbase is still in a pre-alpha state, there are plenty of known issues. If you try things out, please be aware that the following issues are known and documented:

Archetype Query Language

Note: As of now, partial paths will return RM objects in the JSON format used by the Better Platform. Canonical JSON will be supported soonly

- ehr/time_created/value projection is not supported

```
SELECT e/time_created/value FROM EHR e
```

- ehr/ehr_id projection is not supported (instead, an internal ehr/uid is supported)

```
SELECT e/ehr_id FROM EHR e
```

- ehr/uid projection not supported (EHRBASE supports ehr/uid/value but not ehr/uid)

```
SELECT e/uid, e/time_created, e/system_id FROM EHR e
```

- Not supported variables in archetype_id predicates

```
select e/ehr_id/value, e/time_created/value, e/system_id/value from EHR
e CONTAINS COMPOSITION c [$archetype_id]
```

- composition/language projection not supported

```
SELECT c/uid/value, c/name/value, c/archetype_node_id, c/language, c/territory, c/
↪category/value
FROM EHR e [ehr_id/value='dd616472-9432-4004-ad85-fd47affb1cc8'] CONTAINS COMPOSITION␣
↪c
```

- TIMEWINDOW keyword is not supported

```
SELECT e/ehr_id/value FROM EHR e TIMEWINDOW PT12H/2019-10-24
```

Java Client Library

- Occurrences are not recognized (for example events in observations) when auto-generating a dto from an operational template

CHAPTER 2

Getting Started

Warning: WIP

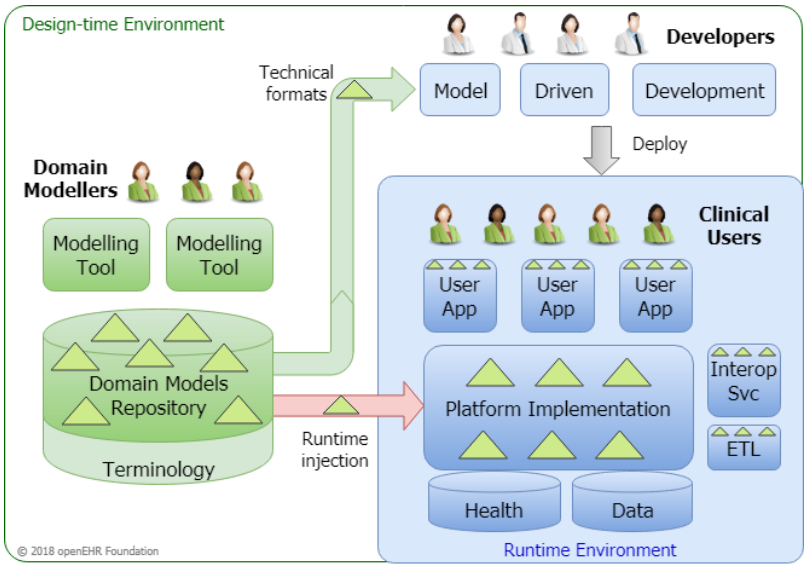
This chapter aims to give an overview an "hello world" example for software developers to build applications based on EHRbase and the openEHR specification. While we use EHRbase as the backend, the example will actually run against every other conformant openEHR implementation that implements the official openEHR REST API.

2.1 openEHR Introduction

Warning: WIP

openEHR is an open platform specification. From a practical perspective you can think about it as an electronic health record that consists of a database that is wrapped with a service layer. The database itself provides only a basic architecture and does not define the clinical content. This is done in a separate modelling layer. Hence, from a developer's perspective, openEHR can be understood as a model-driven software development approach based on an adaptive database that can consume new data definitions at runtime. This allows to manage the high complexity of the medical domain.

As of now, openEHR defines the service access layer based on REST. However, there could be other protocols used in future as the underlying openEHR datamodel is agnostic in terms of API definition. The following figure gives a high-level summary of the approach:



The model-driven openEHR technology ecosystem

Above figure shows the basic concept of separating the clinical definitions on the left side, from the technical implementation inside the platform (which is EHRbase in our case) on the right side.

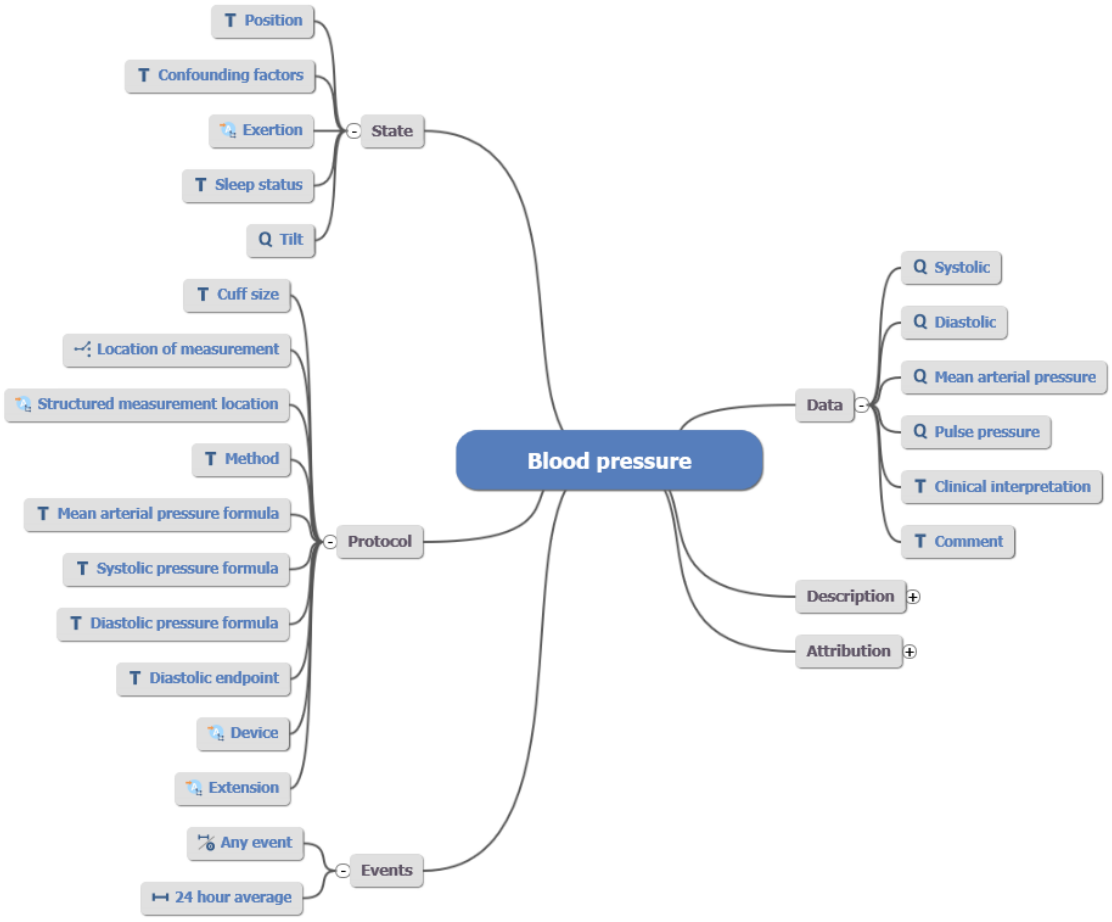
Domain experts define the clinical information models (called Archetypes), which are re-usable models of clinical concepts. Archetypes follow a formalism called Archetype Definition Language, that allows to flexibly model clinical concepts. There are several tools that can be used to create Archetypes in ADL 1.4:

- Archetype Editor
- LinkEHR Editor
- ADL Designer (web only)

The creation of Archetypes is a topic for itself and we will provide another tutorial. Note, that normally system developers should not be too much concerned with the definition and management of Archetypes, as this is the domain of medical information managers and medical professionals.

The goal of Archetypes is to provide standardized sets of data elements and their relations to achieve defined patterns in structured medical documentation. Hence, Archetypes need a strict government to fulfil their potential of enabling semantic interoperability. These models can already contain references to clinical terminologies (e.g. LOINC or SNOMED CT) and particular value sets.

The following image shows the mindmap representation of an Archetype to store data about a blood pressure measurement:



It is obvious that this model is very detailed. This is because Archetypes aim to capture the requirements of different medical specialities. This means, that use-cases from a simple measurement at the general practitioner as well as a detailed assessment through a cardiologist needs to be supported. Normally, the full richness of the model will be reduced before usage in a real-world application.

The governance of Archetypes happens inside a domain model repository. The most commonly used tool is the Clinical Knowledge Manager (CKM)). For international standardization efforts, the CKM is the first address to go to. As local needs cannot be avoided, there are also national instances of the Clinical Knowledge Manager, for example in Germany or Norway

To represent actual clinical use-cases, elements from Archetypes need to be combined inside a Template. You can think of Templates as data sets that can be used to capture data in a form. To create a Template, there are currently two tools available:

- Template Designer
- ADL Designer (web only)

We will soonly add another tutorial to give some more details about the creation of Templates. The Template Designer and the ADL Designer have an export format called Operational Template (OPT). This format is used to inject the use-case specific definitions (that are based on Archetypes) into the openEHR platform (like EHRbase).

This can be done using the POST Template Endpoint of the openEHR REST API.

Now we can take a look at the clinical applications that are based on the openEHR platform. Here, the approaches can differ. The challenge is that the openEHR Reference Model is quite technical a generic to provide optimal handling for computation like validation, storage and querying.

Hence, intermediate formats are often used to make life simpler for developers. In the case of EHRbase, we use the OPT files to enable data-driven development. In the EHRbase Client Library OPTs are used to automatically generate Java classes that can be used to easily build data instances. A data instance in openEHR is called a **composition**.

To allow easier handling, classes are automatically created from the OPT and are much easier for humans to handle. Once data is created, it is transformed to the canonical formats and sent to the openEHR server to a patient's electronic health record. The composition can either be sent alone or as part of a bigger transaction, called a **Contribution**, which can contain different operations on several objects inside the electronic health record, including compositions and folders.

On the server-side, it is checked that all elements inside the composition are valid according to the constraints that were defined in the respective Archetypes and the Template. Once the data has passed all tests, it is permanently stored within a patient's electronic health record. Normally, data can only be updated or logically deleted (in contrast to a physical delete) as electronic health records require a full audit trail about the patient data.

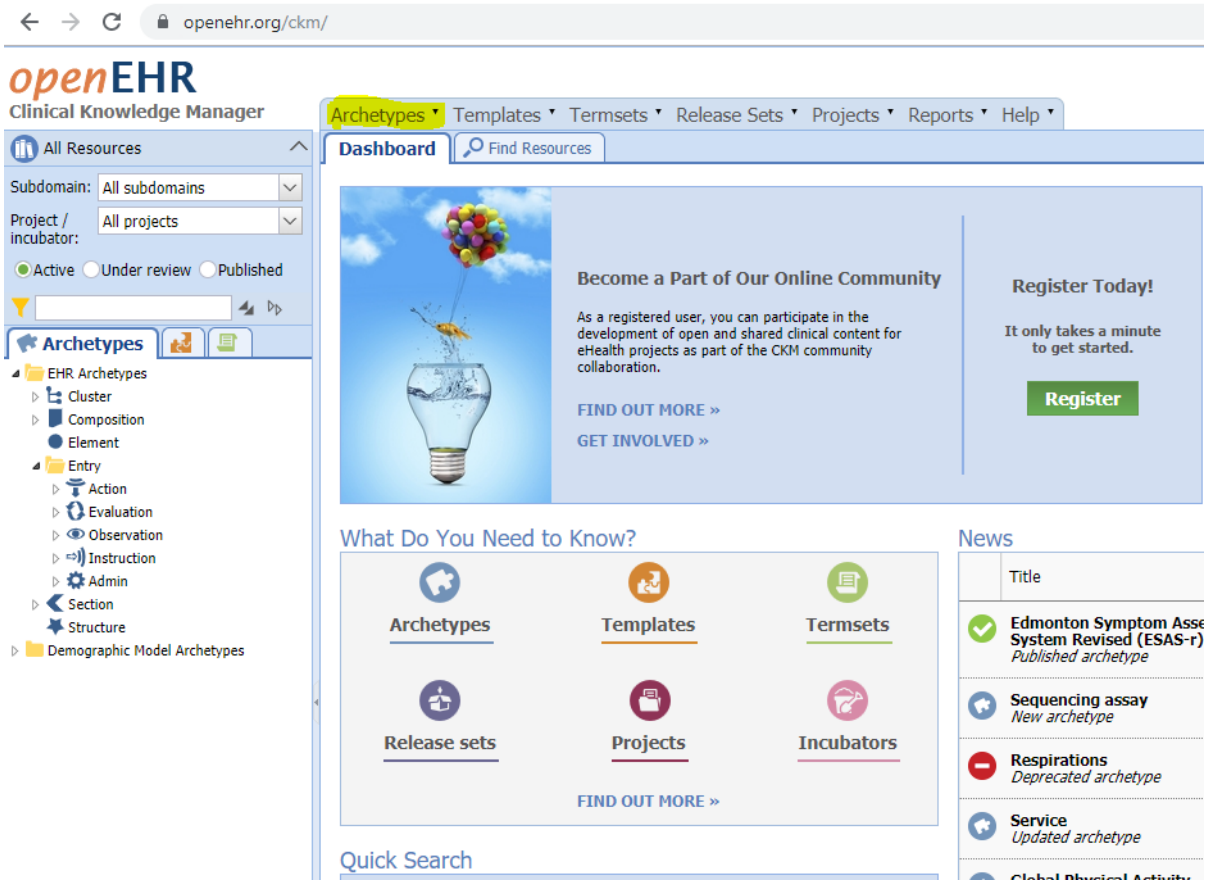
Once the data is stored, it can be retrieved through the openEHR REST API. The most common use-case is to fill user interfaces, for example to plot a list of the latest medications or lab values. This can be done using the Archetype Query Language, a model-based query formalism that only relies on definitions from the Archetypes.

2.2 Step 1: Data Models

Warning: WIP

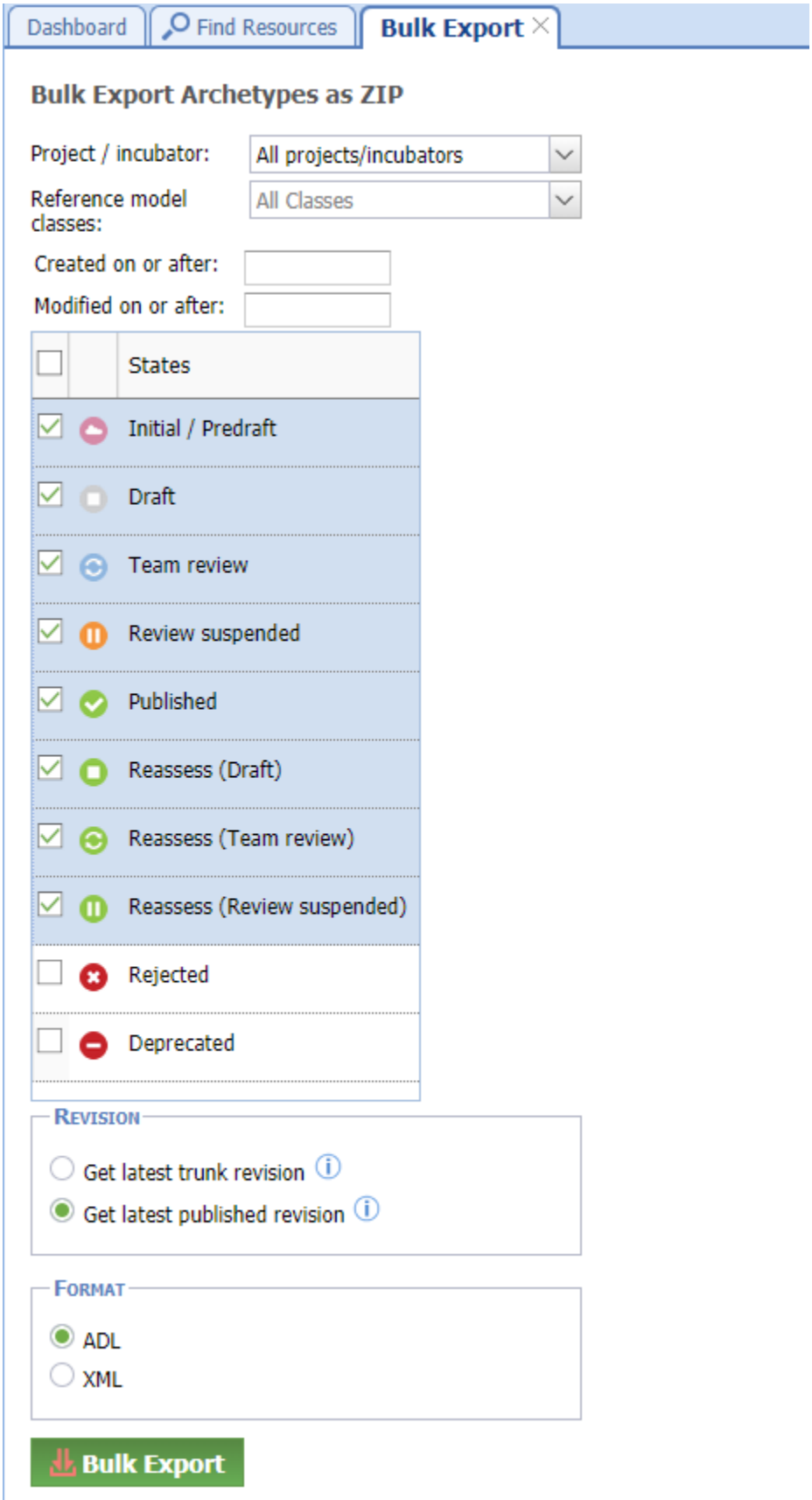
Now, after we got an overview, it's time to put our hands on the tools. Though, if you want to skip this part of the tutorial to directly work with EHRbase, you can get the example files here.

As a first step, we need to obtain the information models. As mentioned in the introduction, the Clinical Knowledge Managers are our first address. To download all Archetypes, go to the International Clinical Knowledge Manager



On the left side, you can find different categories of Archetypes, for example observations that contain data models like blood pressure, body temperature or glasgow coma scale. For our tutorial, we want to get a copy of the archetypes from the Clinical Knowledge Manager.

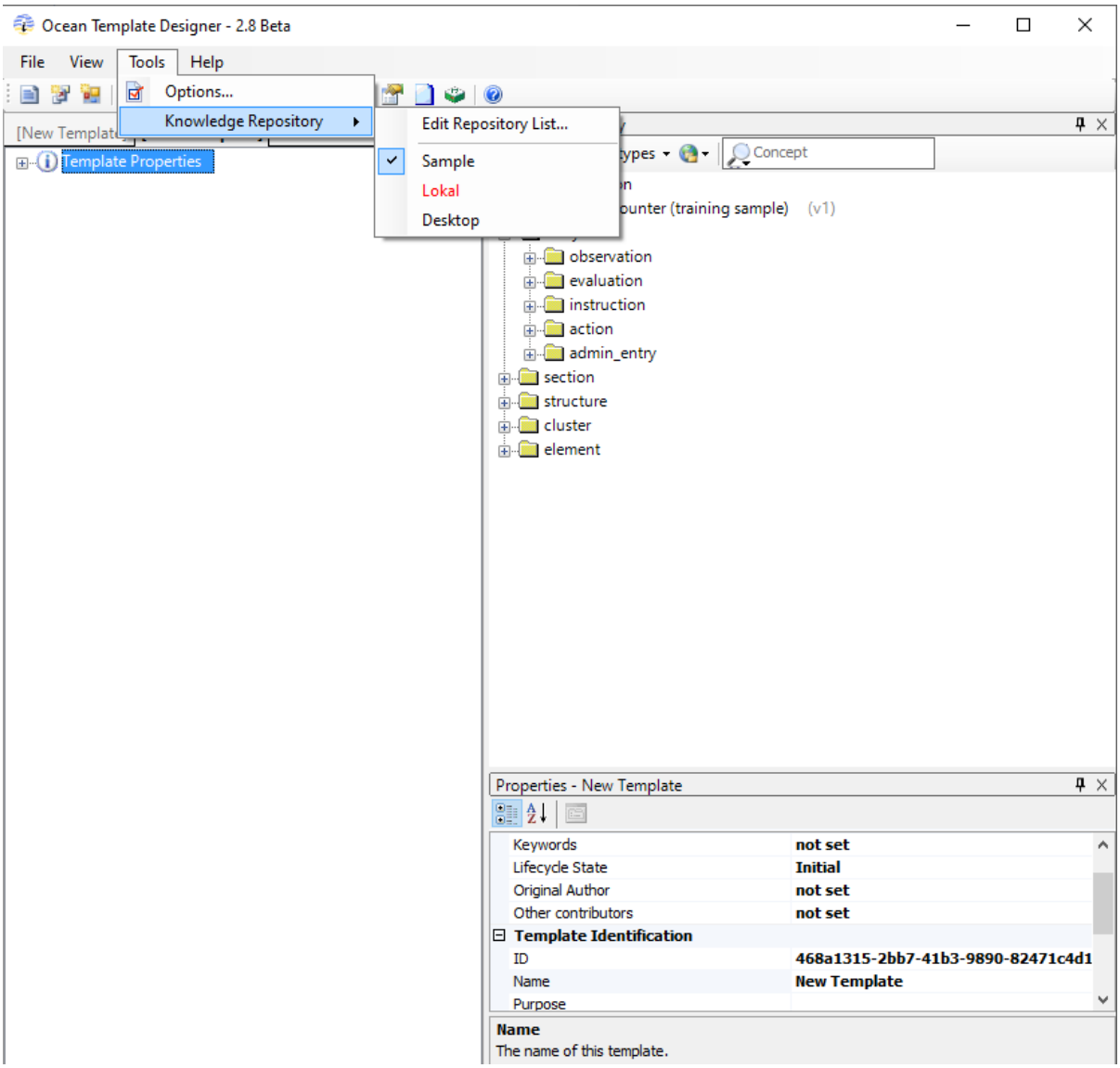
Under Archetypes (marked in the image), you will find a function called *Bulk Export*.


Dashboard
Find Resources
Bulk Export
Bulk Export Archetypes as ZIP
Project / incubator:
All projects/incubators
Reference model classes:
All Classes
Created on or after:
Modified on or after:
States
Initial / Predraft
Draft
Team review
Review suspended
Published
Reassess (Draft)
Reassess (Team review)
Reassess (Review suspended)
Rejected
Deprecated
Revision
Get latest trunk revision
Get latest published revision
Format
ADL
XML
Bulk Export

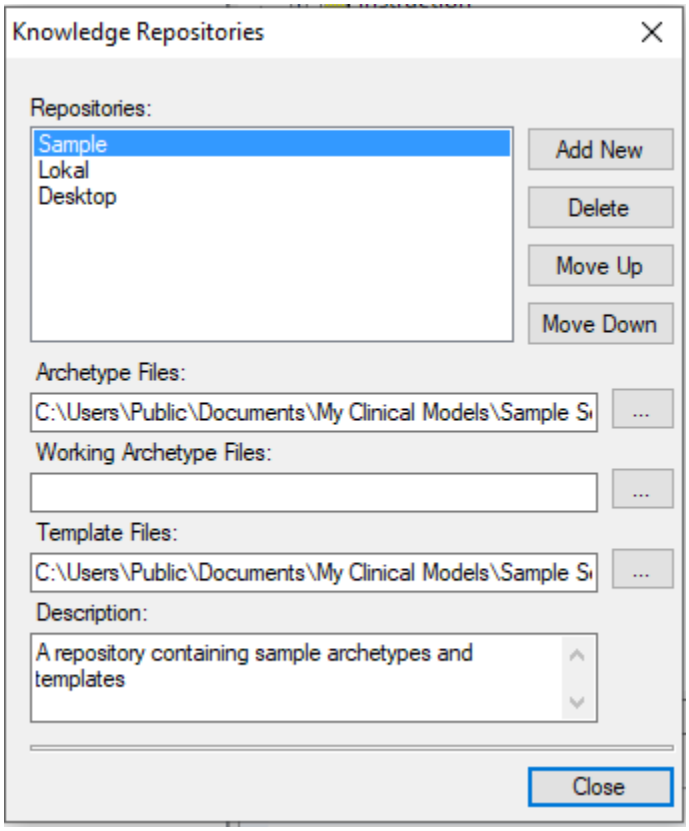
You can choose if the export should only contain Archetypes from a selected project or all and depending on its lifecycle status (published, draft etc.). Choose to get the latest published revision and use ADL (Archetype Definition Language) as export format. Clicking **Bulk Export** will then download a zip folder containing all Archetypes meeting the criteria.

Next, install the Template Designer. The process should be straight forward (At least on Windows). Alternatively, the ADL Designer can also be used to create Templates. However, you will need to upload your files or provide them through a git instance.

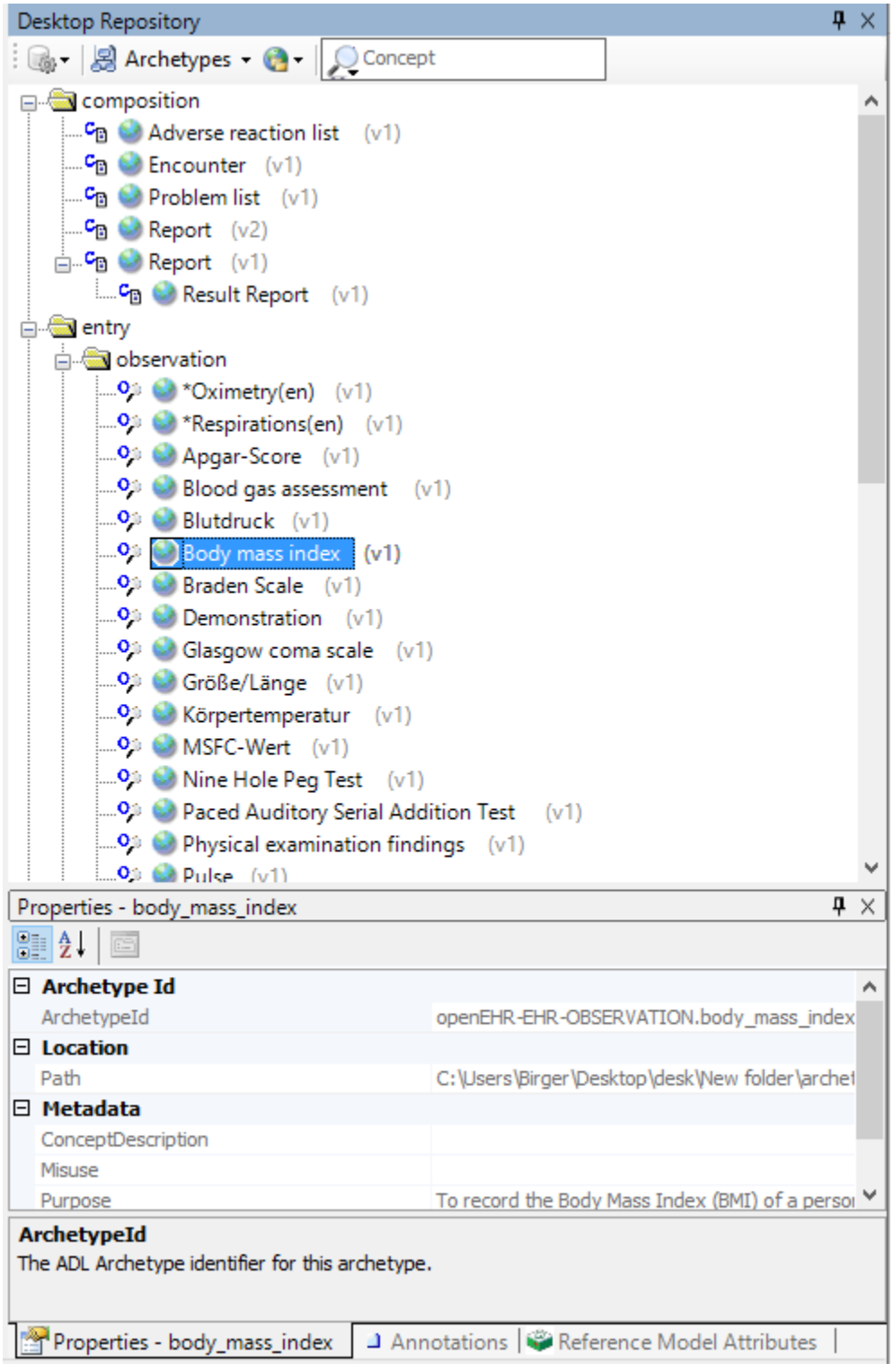
Open the Template Designer. The first step is to configure a *Knowledge Repository*.



Click on *Edit Repository List*

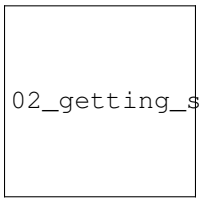


Set *Archetype Files* to the path where you unzipped the Archetypes you obtained through the Bulk Export. When you then select your new repository, the Archetypes should appear on the right window:



Now you can start to create your own Template. Typically, a Template needs an Archetype of type *Composition* as the root element. The *Composition* Archetypes provide the basic structure for the Template through *Slots* (which can be filled with Archetypes) and predefined metadata elements. In our example, we us the *Self Monitoring* Archetype. Just drag and drop the Archetype from the right panel to the left panel. Additionally, we add a blood pressure Archetype. Next, you can define further constraints on the particular elements, for example defining their cardinality, remove single elements, add terminology bindings etc.

We could also fill the slot within the blood pressure Archetype with a device Archetype to collect information about the device used for the measurement.



Finally, give your Template a name. Then you can Export the Template in the Operational Template (OPT) Format (File –> Export –> As Operational Template). This is all you need to upload your Template to EHRbase or any other openEHR server.

2.3 Step 2: Upload a Template

After we created the Template, it's time to put upload it to EHRbase. The data format that is uploaded to an openEHR server like EHRbase is called an Operational Template (OPT-File). You can find an example of an OPT here.

Please see https://specifications.openehr.org/releases/ITS-REST/latest/definitions.html#definitions-adl-1.4-template-post for the REST endpoint that you can use in EHRbase to POST a template. On a local instance of EHRbase, the URL should be like localhost:8080/ehrbase/rest/openehr/v1/definition/template/adl1.4

Copy the content of the OPT file into the body of the REST call. Make sure that the Content-Type attribute is set to application/XML. After the successful call, you should receive a 200 response code.

2.3.1 Client Library

As an alternative to directly using the REST API, the EHRbase Client Library provides functionality to provide a Template.

```
OPERATIONALTEMPLATE template = TemplateDocument.Factory.
↪parse(OperationalTemplateTestData.BLOOD_PRESSURE_SIMPLE.getStream()).getTemplate();
String templateId = "ehrbase_blood_pressure_simple.de.v" + RandomStringUtils.
↪randomNumeric(10);
template.getTemplateId().setValue(templateId);
String actual = new DefaultRestTemplateEndpoint(cut).upload(template);
```

2.4 Step 3: Create an EHR

Warning: WIP

When you start EHRbase from scratch, you will find an empty electronic health record. OpenEHR has a patient-centric architecture. This means that all clinical information inside the database are associated with the EHR of a patient. Hence, the first thing to get started is the creation of an EHR for a patient.

Beware that demographic data of a patient (name, date of birth etc.) are not stored inside an openEHR system by design to ensure a clear separation from the clinical data. Hence, patients are not directly represented in openEHR but their electronic health record. In many cases, a separate demographics service (for example an IHE PIX/PDQ actor, a FHIR Server, an openEHR Demographics Repository or a custom solution) is used.

To create a new EHR, you can either directly use the openEHR REST API or a function within the EHRbase Client Library that encapsulates the REST call.

The REST API https://specifications.openehr.org/releases/ITS-REST/latest/ehr.html#ehr-ehr-post

2.4.1 REST

In this tutorial, we assume that we have a new patient coming to our organization. We simply make a REST call with an empty body.

```
{
  "system_id": {
        "value": "d60e2348-b083-48ce-93b9-916cef1d3a5a"
  },
  "ehr_id": {
        "value": "7d44b88c-4199-4bad-97dc-d78268e01398"
  },
  "ehr_status": {
        "id": {
          "_type": "OBJECT_VERSION_ID",
          "value": "8849182c-82ad-4088-a07f-48ead4180515::openEHRSys.example.com::1"
        },
        "namespace": "local",
        "type": "EHR_STATUS"
  },
  "ehr_access": {
        "id": {
          "_type": "OBJECT_VERSION_ID",
          "value": "59a8d0ac-140e-4feb-b2d6-af99f8e68af8::openEHRSys.example.com::1"
        },
        "namespace": "local",
        "type": "EHR_ACCESS"
  },
  "time_created": {
        "value": "2015-01-20T19:30:22.765+01:00"
  }
}
```

In the result, you should find the EHR ID. This ID will be needed for further operations.

```
"ehr_id": {
        "value": "7d44b88c-4199-4bad-97dc-d78268e01398"
  }
```

2.4.2 Client Library

In the EHRbase Client Library, creating a new EHR object is straight forward:

```
openEhrClient = DefaultRestClientTestHelper.setupDefaultRestClient();
EhrEndpoint ehrEndpoint = openEhrClient.ehrEndpoint();
UUID ehr = ehrEndpoint.createEhr();
```

2.5 Step 4: Load Data

Warning: WIP

Step number 4 brings us to the core functionality of openEHR: creating and storing clinical data! For this purpose, we will re-use the Template that we created in step 1. As data instances in openEHR are stored as instances of its Reference Model, it's rather difficult to read for humans. However, this level of abstraction is needed inside the backend to achieve high scalability.

For application developers, more accessible formats are needed. The EHRbase Client Library allows to use a Template (as OPT) as input and automatically create java classes. These can then be used to create the data. We explain this processes step by step. We assume that you have successfully built the Client Library.

Firstly, you need to create the Java classes from the OPT. This could look like as follows:

```
java -jar client-library-0.2.0.jar -opt "C:\Users\MyUser\Desktop\HiGHmed_Cardio_
↪Monitoring_v1.opt" -out "C:\openEHR SDK\ehrbase_client_
↪library\src\test\java\org\ehrbase\client\classgenerator" -package ""
```

You should find a file named *HiGHmed_Cardio_Monitoring_v1.java* inside your project structure that should look like this:

```
...
@Entity
@Archetype("openEHR-EHR-COMPOSITION.self_monitoring.v0")
@Template("HiGHmed_Cardio_Monitoring.v1")
public class HighmedCardioMonitoringV1 {
 @Path("/context/end_time|value")
 private TemporalAccessor endTimeValue;

 @Path("/language")
 private CodePhrase language;

 @Path("/context/health_care_facility")
 private PartyIdentified healthCareFacility;

 @Path("/composer|external_ref")
 private PartyRef composerExternalref;

...
```

Next, we can create a new test function like this:

```
   public static HighmedCardioMonitoringV1 buildCardioExample(){

    //Create the composition instance and add metadata
    HighmedCardioMonitoringV1 cardioMonitoring = new HighmedCardioMonitoringV1();
    cardioMonitoring.setLanguage(new CodePhrase(new TerminologyId("ISO_639-1"), "de
↪"));
    cardioMonitoring.setTerritory(new CodePhrase(new TerminologyId("ISO_3166-1"), "DE
↪"));
    cardioMonitoring.setSettingDefiningcode(new CodePhrase(new TerminologyId("openehr
↪"), "229"));

    //Create a blood pressure object
```

(continues on next page)

(continued from previous page)

```
    HighmedCardioMonitoringV1.Blutdruck bloodpressure = new HighmedCardioMonitoringV1.
↪Blutdruck();

    //Add data for systolic and diastolic blood pressure
    bloodpressure.setSystolischMagnitude(160d);
    bloodpressure.setSystolischUnits("mm[HG]");

    bloodpressure.setDiastolischMagnitude(120d);
    bloodpressure.setDiastolischUnits("mm[HG]");

    //Add data for a medical device
    HighmedCardioMonitoringV1.Blutdruck.MedizingerT geraet = new␣
↪HighmedCardioMonitoringV1.Blutdruck.MedizingerT();
    geraet.setBeschreibungValue("OMRON Sensor");

    DvIdentifier identifier = new DvIdentifier();
    identifier.setId("4567879799");
    geraet.setEindeutigeIdentifikationsnummerId(identifier);

    List<HighmedCardioMonitoringV1.Blutdruck> bpList = new ArrayList<>();
    bpList.add(bloodpressure);

    cardioMonitoring.setBlutdruck(bpList);

    return cardioMonitoring;
}
```

Finally, the composition can be sent to the openEHR server:

```
CompositionEndpoint compositionEndpoint = openEhrClient.compositionEndpoint(ehr);
UUID compositionId = compositionEndpoint.
↪saveCompositionEntity(highmedCardioMonitoringV1);
```

Congratulations, you stored your first clinical data inside EHRbase! Next, we will take a look how we can retrieve the data using the Archetype Query Language.

CHAPTER 3

Development

This section gives information about the Development of EHRbase and documents the software in detail.

3.1 Developing

> **Warning:** WIP

For the moment, please see the EHRbase GitHub repository for developing information, issue tracker and the source code.

3.2 Technical Documentation

This part of the documentation lists and explains technical details of the implementation of EHRbase.

3.2.1 Overview

> **Warning:** WIP

- openEHR
- REST
- AQL
- etc.

3.2.2 Service Layer

Warning: WIP

General

The service layer of EHRbase is composed of . . .

openEHR Platform Abstract Service Model

Based on the openEHR Platform Abstract Service Model the following check list is build to give an overview and document the current state. Each service component has a table documenting the current state of

- implementation of the *method* itself, if applicable
- implementation and utilization of the *pre checks* of the method, if applicable
- implementation and utilization of the *post checks* of the method, if applicable

Services

EHR

For more details see I_EHR_SERVICE in the official documentation.

Method	Implemented	Pre	Post
has_ehr	Yes	/	/
has_ehr_for_subject	I	/	/
create_ehr	C	/	No
create_ehr_with_id	C	No	No
create_ehr_for_subject	No	/	/
create_ehr_for_subject_with_id	No	No	/
get_ehr	No	No	/
get_ehrs_for_subject	No	/	/
i_ehr	No	/	/

Methods with **I** note are currently indirectly implemented. Their functionality is available, but the general signature might be different.

Methods with **C** note are currently combined in a more general *createEhr* method.

EHR_STATUS

For more details see I_EHR_STATUS the in official documentation.

Method	Implemented	Pre	Post
has_ehr_status_version	I	Yes	/
get_ehr_status	Yes	Yes	/
get_ehr_status_at_time	I	Yes	/
set_ehr_queryable	C	No	No
set_ehr_modifiable	C	No	No
clear_ehr_queryable	C	No	No
clear_ehr_modifiable	C	No	No
update_other_details	C	/	/
get_ehr_status_at_version	Yes	Yes	/
get_versioned_ehr_status	No	No	No

Methods with **I** note are currently indirectly implemented. Their functionality is available, but the general signature might be different.

Methods with **C** note are currently combined in a more general *updateStatus* method.

DIRECTORY

For more details see I_EHR_DIRECTORY the in official documentation.

Method	Implemented	Pre	Post
has_directory			
has_path			
create_directory			
get_directory			
get_directory_at_time			
update_directory			
delete_directory			
has_directory_version			
get_directory_at_version			
get_versioned_directory			

COMPOSITION

For more details see I_EHR_COMPOSITION the in official documentation.

Method	Implemented	Pre	Post
has_composition			
get_composition_latest			
get_composition_at_time			
get_composition_at_version			
get_versioned_composition			
create_composition			
update_composition			
delete_composition			

CONTRIBUTION

For more details see I_EHR_CONTRIBUTION the in official documentation.

Method	Implemented	Pre	Post
has_contribution			
get_contribution			
commit_contribution			
list_contributions			
contribution_count			

3.2.3 New Contain Clause Resolution Strategy

C. Chevalley 3.7.20

3.2.4 Backgroud

AQL specifies the important clause 'CONTAINS'. This allows to specify a containment criteria on specified archetypes anywhere into composition projections. The specification is found in openEHR AQL containment. As mentioned in the specification, 'CONTAINS' specifies an *hierarchical* relationship with the Tree based data architecture (hence not to be confused with a WHERE clause criteria). Hierarchical constraint is modelized using connected and acyclic graph; a node can be accessed from the root through a unique path.

Previous Approach

The previous strategy was based on maintaining a specific containment table based on a hierarchical data representation using PostgreSQL ltree. The algorithm was based on identified AQL paths during the composition serialization: each path expression was then stored in a simplify way as to describe the hierarchy of archetypes within the composition, this for each composition. The table was then used to build the SQL expression corresponding to an AQL statement:

- identify the template(s) matching the contain clause
- retrieve the path for a given contain constraint for each identified template(s)

The resulting SQL expression is a combination (UNION) of SQL statement for each template.

An example of containment records is as follows:

```
CONTAINS COMPOSITION c CONTAINS OBSERVATION o [openEHR-EHR-OBSERVATION.
pulse-oximetry.v1]
```

Is translated as

```
SELECT composition_id FROM ehr.contain WHERE label ~= '*.openEHR_EHR_OBSERVATION_
↪pulse_oximetry_v1'
```

The template Id is then retrieve from the correlation between the composition entry (ehr.entry) and the template_id attribute. The same logic is used to retrieve the path of a particular node relatively to a template.

Although this approach was initially satisfactory, it has been seen as impacting performance whenever the number of records increases. As shown in the above example, the number of entries for a single composition can be significant and, in the lack of proper indexing, the identification of a template may require costly sequential search. Further, the construction of an SQL expression corresponding to an AQL CONTAINS clause was problematic. Another issue was that item_structure in /context/other_context was not referenced in containment and then was not resolved for querying.

New Approach

Assumptions

This approach assumes that all stored compositions are bound to one known template (at the time of this writing, operational template v1.4). A template is known whenever it is defined in the platform, it is stored in the DB in table `ehr.template_store`

Objectives

The new logic consists in resolving an AQL CONTAINS clause by:

- identifying the template(s) matching the constraints
- resolving the paths for the nodes defined in the CONTAINS clause

Identified templates are used to build the resulting SQL expression, each identified template produces a SQL query. At the end of the process, SQL queries are chained by a UNION clause.

Resolved paths are used to construct the json path expression used to query JSONB structure in the DB.

Technical Approach

Operational Template Traversal

All resolution are now based on so-called WebTemplates (class `OptVisitor`) providing a tree construct detailing all constraints and attributes of an operational template. The tree structure is traversed using JsonPath expressions (see f.e. Baeldung's guide on this).

For instance, to check the existence of a node containment and return the corresponding AQL path, the following logic is illustrated as follows.

Assume we want to retrieve the template(s) where the following expression is satisfied:

```
contains COMPOSITION c[openEHR-EHR-COMPOSITION.report-result.v1] contains
CLUSTER f [openEHR-EHR-CLUSTER.case_identification.v0]
```

The corresponding jsonpath expression to traverse the WebTemplate is:

```
$..[?(@.node_id == 'openEHR-EHR-COMPOSITION.report-result.v1')]..[?(@.node_id
== 'openEHR-EHR-CLUSTER.case_identification.v0')]
```

When applied to template `Virologischer Befund`, the following structure is returned (these are the attributes for the retrieved node)


```
{
    "min" : "1",
    "aql_path" : "/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_
↪identification.v0]",
    "max" : "1",
    "children" : " size = 2",
    "name" : "Fallidentifikation",
    "description" : "Zur Erfassung von Details zur Identifikation eines Falls im␣
↪Gesundheitswesen.",
    "id" : "fallidentifikation",
    "type" : "CLUSTER",
    "category" : "DATA_STRUCTURE",
```


(continues on next page)

(continued from previous page)

```
    "node_id" : "openEHR-EHR-CLUSTER.case_identification.v0",
}
```

The corresponding AQL path for node `openEHR-EHR-CLUSTER.case_identification.v0` in template `Virologischer Befund` is `/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_identification.v0]`

The corresponding WebTemplate section for this particular node is represented as follows:

```
{
  "min": 1,
  "aql_path": "/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_
↪identification.v0]",
  "max": 1,
  "children": [
    {
      "min": 1,
      "aql_path": "/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_
↪identification.v0]/items[at0001]",
      "max": 1,
      "name": "Fall-Kennung",
      "description": "Der Bezeichner/die Kennung dieses Falls.",
      "id": "fall_kennung",
      "category": "ELEMENT",
      "type": "DV_TEXT",
      "constraints": [
        {
          "aql_path": "/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_
↪identification.v0]/items[at0001]/value",
          "mandatory_attributes": [
            {
              "name": "Value",
              "attribute": "value",
              "id": "value",
              "type": "STRING"
            }
          ],
          "attribute_name": "value",
          "constraint": {
            "occurrence": {
              "min": 1,
              "max_op": "\u003c\u003d",
              "min_op": "\u003e\u003d",
              "max": 1
            }
          },
          "type": "DV_TEXT"
        }
      ],
      "node_id": "at0001"
    },
    {
      "aql_path": "/context/other_context[at0001]/items[openEHR-EHR-CLUSTER.case_
↪identification.v0]/items",
      "name": "Items",
      "attribute": "items",
      "id": "items",
```

(continues on next page)

(continued from previous page)

```
      "occurrence": {
        "min": 1,
        "max_op": "\u003c\u003d",
        "min_op": "\u003e\u003d",
        "max": 1
      },
      "category": "ATTRIBUTE",
      "type": "ITEM"
    }In other terms, t
  ],
  "name": "Fallidentifikation",
  "description": "Zur Erfassung von Details zur Identifikation eines Falls im␣
↪Gesundheitswesen.",
  "id": "fallidentifikation",
  "type": "CLUSTER",
  "category": "DATA_STRUCTURE",
  "node_id": "openEHR-EHR-CLUSTER.case_identification.v0"
},
```

Whenever the node_id is not specified, the jsonpath expression uses class names. For example the following AQL

```
SELECT location FROM EHR e CONTAINS COMPOSITION CONTAINS ADMIN_ENTRY CONTAINS
location [openEHR-EHR-CLUSTER.location.v1]
```

Is translated as:

```
$..[?(@.type == 'COMPOSITION')]..[?(@.type == 'ADMIN_ENTRY')]..[?(@.node_id ==
'openEHR-EHR-CLUSTER.location.v1')]
```

AQL Clause Interpretation

Contains clause interpretation consists in parsing the AQL expression (ANTLR) and create a corresponding list of propositions to evaluate.

The logic is based on the recursive traversal of the tree expression (AST), from bottom left to the top of the tree, and create the template traversal query as well as the boolean validations if any if the expression contains logical operators (AND, OR, XOR . . .).

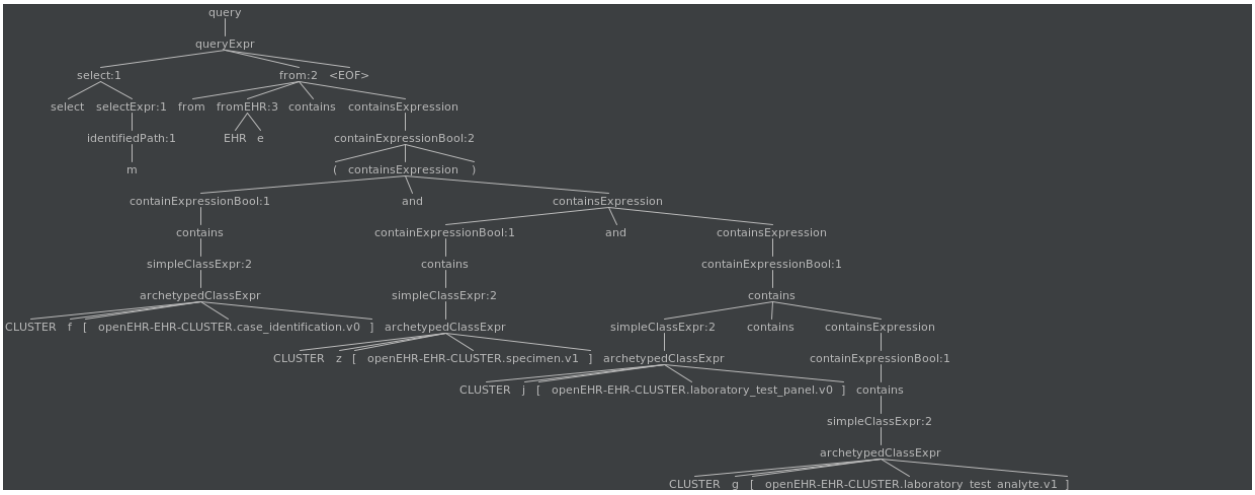
The evaluation does check first simple containment chains (CONTAINS. . . CONTAINS. . . CONTAINS. . .) using WebTemplate traversals described above, and then checks the logical propositions based on these.

Example

AQL expression:

```
select
m
from EHR e
contains (
    CLUSTER f[openEHR-EHR-CLUSTER.case_identification.v0] and
    CLUSTER z[openEHR-EHR-CLUSTER.specimen.v1] and
    CLUSTER j[openEHR-EHR-CLUSTER.laboratory_test_panel.v0]
    contains CLUSTER g[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1])
```

The containments are evaluated with the following tree



The containments are evaluated as follows:

1. "**CLUSTERf[openEHR-EHR-CLUSTER.case_identification.v0]**" -
2. "**CLUSTERz[openEHR-EHR-CLUSTER.specimen.v1]**"
3. "**CLUSTERg[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1]**" as in `CLUSTER j[openEHR-EHR-CLUSTER.laboratory_test_panel.v0] contains CLUSTER g[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1]`)
4. "**CLUSTERz[openEHR-EHR-CLUSTER.specimen.v1] and CLUSTERj[openEHR-EHR-CLUSTER.laboratory_test_panel.v0]containsCLUSTERg[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1]**": check the INTERSECTION of the results from 2 AND 3 above
5. "**CLUSTERf[openEHR-EHR-CLUSTER.case_identification.v0] and CLUSTERz[openEHR-EHR-CLUSTER.specimen.v1]andCLUSTERj[openEHR-EHR-CLUSTER.laboratory_test_panel.v0]containsCLUSTERg[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1]**": check the INTERSECTION of the results from 1 & 4
6. "**(CLUSTERf[openEHR-EHR-CLUSTER.case_identification.v0]andCLUSTERz[openEHR-EHR-CLUSTER.specimen.v1]andCLUSTERj[openEHR-EHR-CLUSTER.laboratory_test_panel.v0]containsCLUSTERg[openEHR-EHR-CLUSTER.laboratory_test_analyte.v1])**": same as 5 since it is enclosed in parenthesis.

If another operator is used: OR or XOR, then we apply UNION or DISJUNCTION respectively.

DB Changes

The two most significant changes are

1. Deprecation of table ehr.containment. This table is now removed, as well as all logic associated to its population.
2. New encoding of composition entry (item_structure)

The composition entry encoding (jsonb) has now the composition name encoded outside the json structure as a dv_coded_text (UDT) in table ehr.entry and removed from the archetype node id in the composition path.

This change is required since now the identified path is a generic AQL path without composition dependent values.

```
{
  "/name": [
    {
      "value": "Bericht"
```

(continues on next page)

(continued from previous page)

```
    }
  ],
  "/$CLASS$": "Composition",
  "/composition[openEHR-EHR-COMPOSITION.report.v1 and name/value='Bericht']": {
    "/content[openEHR-EHR-OBSERVATION.blood_pressure.v2]": [
      {
        "/name": [
          {
            "value": "Blutdruck"
          }
        ],
        "/$CLASS$": "Observation"
}
```

The name/value attribute in the node id is now passed as an external attribute 'name' and the composition item_structure is encoded as

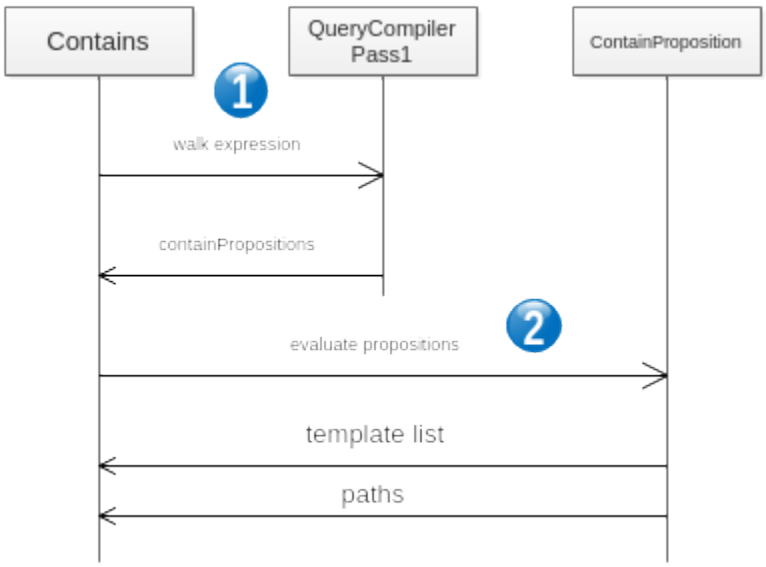
```
{
  "/name": [
    {
      "value": "Bericht"
    }
  ],
  "/$CLASS$": "Composition",
  "/composition[openEHR-EHR-COMPOSITION.report.v1]": {
    "/content[openEHR-EHR-OBSERVATION.blood_pressure.v2]": [
      {
        "/name": [
          {
            "value": "Blutdruck"
          }
        ],
        "/$CLASS$": "Observation
}
```

While name is

```
(Bericht,,,,)
```

Processing

The sequence of containment resolution is the following



1. Consists in parsing the AQL CONTAINS expression and build the propositions as described above.
2. The propositions are evaluated as
    1. Simple containment chains using cached WebTemplates
    2. Computed boolean expressions based on the simple containment chains

Further Enhancements

1. At this stage, ehr_status/other_details is not part of the contains resolution. The main issue here is that it is generally not associated to a valid template.
2. There need to do more research for archetype_slots in a ANY type.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search