SensorThings deployment, configuration and maintenance documentation

|  |  |
| --- | --- |
| Author: | Katharina Schleidt |
| Date: | 21.06.2018 |

#### **Contents**

[1. Introduction 5](#_Toc517438352)

[2. Required Software 6](#_Toc517438353)

[3. Database Configuration 6](#_Toc517438354)

[3.1. Set Postgres Password 6](#_Toc517438355)

[3.2. Create sensorthings User 6](#_Toc517438356)

[3.3. Create a sensorthings database 7](#_Toc517438357)

[4. SensorThings API Deployment & Configuration on Tomcat 7](#_Toc517438358)

[4.1. SensorThings API Deployment 7](#_Toc517438359)

[4.2. Postgres Jar Files 8](#_Toc517438360)

[4.3. SensorThings API Configuration 8](#_Toc517438361)

[4.3.1. Database Settings 8](#_Toc517438362)

[4.3.2. Service URL 9](#_Toc517438363)

[4.3.3. Default and Max Top 9](#_Toc517438364)

[4.3.4. Identifier Types 9](#_Toc517438365)

[4.3.5. Extended Properties Section & String IDs 10](#_Toc517438366)

[5. BRGM Rawdata import 10](#_Toc517438367)

[5.1. Preparation Rawdata for URI-IDs 11](#_Toc517438368)

[6. View Creation 11](#_Toc517438369)

[6.1. Additional Tables and Materialized Views 11](#_Toc517438370)

[6.1.1. TABLE rawdata.proc\_uri 11](#_Toc517438371)

[6.1.2. MATERIALIZED VIEW rawdata.obsinfo 12](#_Toc517438372)

[6.2. FROST Materialized Views 13](#_Toc517438373)

[6.2.1. MATERIALIZED VIEW DATASTREAMS 13](#_Toc517438374)

[6.2.2. MATERIALIZED VIEW FEATURES 14](#_Toc517438375)

[6.2.3. MATERIALIZED VIEW OBSERVATIONS 15](#_Toc517438376)

[6.2.4. MATERIALIZED VIEW OBS\_PROPERTIES 15](#_Toc517438377)

[6.2.5. MATERIALIZED VIEW SENSORS 16](#_Toc517438378)

[6.2.6. MATERIALIZED VIEW THINGS 16](#_Toc517438379)

[6.2.7. MATERIALIZED VIEW LOCATIONS 17](#_Toc517438380)

[6.2.8. MATERIALIZED VIEW THINGS\_LOCATIONS 17](#_Toc517438381)

[6.3. FROST Materialized Views URI-ID 18](#_Toc517438382)

[6.3.1. MATERIALIZED VIEW DATASTREAMS 18](#_Toc517438383)

[6.3.2. MATERIALIZED VIEW FEATURES 19](#_Toc517438384)

[6.3.3. MATERIALIZED VIEW OBSERVATIONS 20](#_Toc517438385)

[6.3.4. MATERIALIZED VIEW OBS\_PROPERTIES 21](#_Toc517438386)

[6.3.5. MATERIALIZED VIEW SENSORS 22](#_Toc517438387)

[6.3.6. MATERIALIZED VIEW THINGS 22](#_Toc517438388)

[6.3.7. MATERIALIZED VIEW LOCATIONS 23](#_Toc517438389)

[6.3.8. MATERIALIZED VIEW THINGS\_LOCATIONS 24](#_Toc517438390)

[6.4. FROST Tables URI-ID 24](#_Toc517438391)

# Introduction

During the previous years, BRGM has deployed interoperable OGC services on top of the French Groundwater Information Network (GIN); this task was done in close alignment with the requirements stemming from INSPIRE, specifically those pertaining to the provision of Environmental Monitoring Facilities/Network descriptions together with their associated observational data. As a result, a coherent system of linked resources are made available, and served using OGC WFS and SOS.

In parallel to this existing work, the OGC has been developing additional standards in alignment with new technological developments, including:

* The OGC SensorThings API merges the many of the capabilities of more traditional web services such as WFS and SOS with lightweight data models encoded in JSON as well as a RESTful interface based on the OASIS oData standard.
* The OGC Environmental Linked Features Interoperability Experiment (ELFIE) is working on determining what interlinkages between domain features would best serve to put both these features as well as the observational data linked to them in context, allowing for faster and easier discovery, access and use of the available data.

These developments are of special interest for the further development of the GIN, as they have the potential to make existing data far more accessible, as well as allowing for integration with linked data approaches.

For a first proof-of-concept, the following datasets have been selected as representative of wider GIN data holdings:

A°/ Groundwater Level Monitoring Facilities: Piezometers exposed according to INSPIRE – Environmental Monitoring Facilities theme

B°/ Raw sensor observations pushed by sensors on the field

C°/ Raw Geology logs: compliant to OGC GroundWaterML2 GW\_GeologyLogCoverage

These datasets are presently stored in a PostGreSQL/GIS database, whereby the underlying data model is closely aligned with the relevant OGC and INSPIRE standards, foremost the Environmental Monitoring Facilities (EF) data specification, the INSPIRE Guidance document on the use of the Observations & Measurements (O&M) standard within INSPIRE (D2.9), as well as the underlying OGC standards including O&M and GML.

In order to utilize the SensorThings API within the constraints of INSPIRE, in addition to assuring that all data requirements stemming from the data specifications are fulfilled, the service interface must also be analyzed to determine if all requirements for INSPIRE download services have been met; the INSPIRE Technical Guidance for implementing download services using the OGC Sensor Observation Service and ISO 19143 Filter Encoding will serve as a guideline for this task. Within the ELFIE context, the GeoJSON and JSON-LD standards shall be taken into account, including the OGC Testbed-12 JSON and GeoJSON User Guide. In addition, W3C recommendations on data on the web will be taken into account.

This document details the steps necessary for the implementation of a SensorThings API server (Frauenhofer IOSB implementation) on top of the GIN data holdings.

# Required Software

In order to deploy the Frauenhofer implementation of the SensorThings API, the following software must first be installed on the Linux server:

**PostgreSQL with PostGIS:** PostgreSQL should be V9.5 or higher; on the demonstrator, PostgreSQL V9.6 has been utilized, together with the PostGIS extension 2.4.1 r16012

**Tomcat:** a Tomcat application server is required for the deployment of the SensorThings API; on the demonstrator, Tomcat V8.5.14 has been deployed.

**SensorThings API:** The current version of the SensorThings API must be downloaded from the FrauenhoferIOSB Github at <https://github.com/FraunhoferIOSB/SensorThingsServer>. As the demonstrator supports the extended properties, this version was deployed.

# Database Configuration

## Set Postgres Password

Set a password for the postgres user.

1. Run the psql command from the postgres user account: sudo -u postgres psql postgres
2. Set the password: \password postgres
3. Enter a password.
4. Close psql.

## Create sensorthings User

To create the sensorthings user via pgadmin3, perform the following steps:

1. Right-click Login Roles -> New Login Role
2. Enter a name (sensorthings)
3. Enter a password (twice)
4. Click OK

CREATE USER sensorthings WITH

LOGIN

NOSUPERUSER

INHERIT

CREATEDB

CREATEROLE

REPLICATION;

## Create a sensorthings database

In order to create the sensorthings database via pgadmin3, perform the following steps:

1. Right-click Databases -> new Database
2. Give a name (sensorthings)
3. Set owner to "sensorthings" (if that's what you called the user in step 3.2)
4. Do not set a Template
5. Click ok

Then, navigate to database sensorthings as user postgres and initialise PostGIS on the newly created database as follows:

1. With the database sensorthings selected, click the SQL button in the menu bar
2. Enter the query CREATE EXTENSION postgis;
3. Click the green "run" button

# SensorThings API Deployment & Configuration on Tomcat

## SensorThings API Deployment

Download the SensorThings API war file from GitHub and copy the SensorThingsServer-x.x.war to the tomcat webapps folder. It's best to rename the war to just SensorThingsServer.war to make upgrading easier when a new version is release (for the demonstrator, the war file was renamed to SensorThingsServer-1.0\_BRGM.war to allow for a separate BRGM instance in addition to the base SensorThings server available as SensorThingsServer-1.0).

Once the SensorThings server has been deployed, but before taking it into operation, various configuration settings must first be set within the configuration file, available in the application directory under META-INF/context.xml. In addition, jar files for Postgres must be provided as described below.

## Postgres Jar Files

Before the SensorThings API can be deployed on Tomcat, two Postgres Jar files must be copied to the Tomcat lib directory to avoid conflicts. Download the following jar files and copy to the $CATALINA\_HOME/lib directory:

Postgres JDBC jar: <http://repo.maven.apache.org/maven2/org/postgresql/postgresql/9.4.1212/postgresql-9.4.1212.jar>

PostGIS jar: <http://repo.maven.apache.org/maven2/net/postgis/postgis-jdbc/2.2.1/postgis-jdbc-2.2.1.jar>

Note: in some cases, this can cause conflicts with other applications deployed on the same TomCat instance (i.e. GeoServer). In such cases, the jar files should be copied to the WEB-INF/lib directory of the SensorThings webapp, so as to be available only to this application.

## SensorThings API Configuration

### Database Settings

Attributes of the Resource tag are used to configure access to the database. In this entry, the URL for the database can be set, as well as the user name and password to be used for the connection. These should be aligned with the database configuration described above.

<Resource

name="***jdbc/sensorthings\_brgm***" auth="Container"

type="javax.sql.DataSource" driverClassName="org.postgresql.Driver"

**url="jdbc:postgresql://localhost:5432/sensorthings\_brgm"**

**username="sensorthings" password="sensorthings"**

maxTotal="20" maxIdle="10" maxWaitMillis="-1"

defaultAutoCommit="false"

/>

In cases where multiple instances of FROST are deployed on the same Tomcat, specific names must be provided for each database and FROST installation. The name specified for the jndi datasource, set to “jdbc/sensorthings\_brgm" in the example below, must also be provided as the resource name in the database settings described above

<Parameter name="persistence.db\_jndi\_datasource" value="***jdbc/sensorthings\_brgm***" description="JNDI data source name"/>

### Service URL

In order to allow the SensorThings server to provide the correct URLs for access to the various classes served, the root service URL must be provided within a Parameter tag as the value for the entry with name="serviceRootUrl".

<Parameter name="serviceRootUrl" **value="http://5.189.139.50:8080/SensorThingsServer-1.0\_BRGM"** description="The base URL of the SensorThings Server without version."/>

### Default and Max Top

It is possible to configure the number of results returned per request, both as a default value as well as from within the request. The following entries from the context.xml file provide the configuration to the FROST Server.

Default:

<Parameter name="defaultTop" value="20" description="The default value for the $top query option."/>

Maximum:

<Parameter name="maxTop" value="100" description="The maximum allowed value for the $top query option."/>

### Identifier Types

In the default configuration, FROST utilizes identifiers of type Long; identifiers are automatically generated for new entries when FROST is running on database tables. However, other identifier types are possible; the following types can be configured:

* de.fraunhofer.iosb.ilt.sta.persistence.postgres.longid.PostgresPersistenceManagerLong
* de.fraunhofer.iosb.ilt.sta.persistence.postgres.stringid.PostgresPersistenceManagerString
* de.fraunhofer.iosb.ilt.sta.persistence.postgres.uuidid.PostgresPersistenceManagerUuid

The configuration entry for setting the identifier type is as follows.

<Parameter override="false"  
name="persistence.persistenceManagerImplementationClass"  
value="de.fraunhofer.iosb.ilt.sta.persistence.postgres.stringid.PostgresPersistenceManagerString"  
description="The java class used for persistence (must implement PersistenceManager interface)"  
/>

### Extended Properties Section & String IDs

Please note that while the extended functionalities (properties on all classes, String IDs) are not yet formally included within the OGC SensorThings API standard, they have been proposed for the upcoming 1.1 release of this standard. Due to popular demand, this functionality is already included within the standard branch of FROST.

## Generate DB

Normally, the SensorThings utilizes its own tables for storage of and access to data. Alternatively, these tables can be modelled as (materialized) views; the creation of these views is described in the section below on View Creation.

If the SensorThings server is to utilize dedicated tables, the server must be tasked to generate the necessary database tables. To do this, access the base URI of the SensorThings server (without the version number “V1.0). On this page, access the link “Database Status and Update”; this will lead you to a page displaying all SQL statements required to create the necessary tables. On this page, click the button “Do Update”.

# INSPIRE Requirements

For the utilization of the SensorThings API within INSPIRE, care must be taken to assure that all data requirements stemming from the relevant INSPIRE Theme data specification, in this case Environmental Monitoring Facilities, must be provided. As not all INSPIRE requirements can be aligned directly to attributes of the SensorThings API classes, the additional data must be provided utilizing the extended properties section. A basic overview of the fields required within the properties sections of the various classes is provided below. As far as this information was available, it has been configured within the views for the SensorThings Server.

For more information, please see the paper **Extending INSPIRE to the Internet of Things through SensorThings API** A. Kotsev et al; Geosciences 2018, 8(6), 221; doi:10.3390/geosciences8060221

**THING**

* PROPERTIES/namespace
* PROPERTIES/mediaMonitored
* PROPERTIES/measurementRegime
* PROPERTIES/mobile
* PROPERTIES/measurementRegime
* PROPERTIES/mobile

**DATASTREAMS**

* PROPERTIES/processType
* PROPERTIES/resultNature

**SENSORS**

* PROPERTIES/responsibleParty
* PROPERTIES/responsibleParty/individualName
* PROPERTIES/responsibleParty/organisationName
* PROPERTIES/responsibleParty/positionName
* PROPERTIES/responsibleParty/language
* PROPERTIES/responsibleParty/adminUnit
* PROPERTIES/responsibleParty/script
* PROPERTIES/responsibleParty/locatorDesignator
* PROPERTIES/responsibleParty/postCode
* PROPERTIES/responsibleParty/electronicMailAddress
* PROPERTIES/responsibleParty/telephoneVoice
* PROPERTIES/responsibleParty/website
* PROPERTIES/responsibleParty/role

**FEATURES**

* PROPERTIES/sampledFeature

# BRGM Rawdata import

The rawdata dump provided by BRGM was dependent on a schema with postgis extension being named “postgis”; in the sensorthings installation, the schema with postgis extension is “public”.

To import the BRGM rawdata into the sensorthings db, a dump was created with the postgis dependency altered to “public”

In addition, two roles must be prepared in the DB in order to import the rawdata dump. These roles must be available as follows.

CREATE USER observations\_p WITH

LOGIN

NOSUPERUSER

INHERIT

CREATEDB

CREATEROLE

REPLICATION;

CREATE USER observations\_reader WITH

LOGIN

NOSUPERUSER

INHERIT

CREATEDB

CREATEROLE

REPLICATION;

## Preparation Rawdata for URI-IDs

Before the views for the URI-ID version were created, some cleaning of the rawdata dump was performed. This pertained to the table FoI2, where a dedicated samplingFeature was provided for each FoI. As many of these were redundant (same description, identifier and localidentifier values, though different localidentifier\_save values), these were reduced to one version, and the sampledFeature entry linking from the FoI adjusted accordingly.

After the data was cleaned, a baseid field was added to the tables foi2, observation and result. Where a localidentifier was provided within foi2, this values was taken; in cases where no value was provided for localidentifier, the value from localidentifier\_save was used instead. In addition, slashes “/” contained in the localidentifier were replaced by dashes “-“ for easier handling. These string identifiers were then propagated from the FoI entries to the tables observation and result, for utilization in the creation of the URI Identifiers via the views.

# View Creation

Before the views can be created, the existing sensor things tables must be renamed. In the demonstrator implementation, all existing sensor things tables have been renamed with a “\_T” suffix, to differentiate them from the views. In addition, a table as well as a materialized view was created in order to provide additional information not directly contained within the rawdata dump.

Please note that the following tables pertaining to multi-datastreams and historical locations have not been created, as they are not of relevance for the current data holdings:

* HIST\_LOCATIONS
* LOCATIONS\_HIST\_LOCATIONS
* MULTI\_DATASTREAMS
* MULTI\_DATASTREAMS\_OBS\_PROPERTIES

## Additional Tables and Materialized Views

### TABLE rawdata.proc\_uri

The table rawdata.proc\_uri was created in order to provide information on the measurement procedure as required by INSPIRE. The additional information was accessed from the codelists available at <http://id.eaufrance.fr>

The table rawdata.proc\_uri is structured as follows:

CREATE TABLE rawdata.proc\_uri

(

id character varying(200) COLLATE pg\_catalog."default" NOT NULL,

namespace character varying(200) COLLATE pg\_catalog."default",

type character varying(200) COLLATE pg\_catalog."default",

cit\_id character varying(200) COLLATE pg\_catalog."default",

cit\_name character varying(500) COLLATE pg\_catalog."default",

cit\_shortname character varying(200) COLLATE pg\_catalog."default",

cit\_date date,

cit\_link character varying(200) COLLATE pg\_catalog."default",

rp\_org character varying(200) COLLATE pg\_catalog."default",

rp\_lang character varying(10) COLLATE pg\_catalog."default",

rp\_country character varying(200) COLLATE pg\_catalog."default",

rp\_script character varying(10) COLLATE pg\_catalog."default",

id\_raw character varying(200) COLLATE pg\_catalog."default",

CONSTRAINT proc\_uri\_pkey PRIMARY KEY (id)

);

### MATERIALIZED VIEW rawdata.obsinfo

The materialized view rawdata.obsinfo was created in order to provide information on the feature of interest as required by INSPIRE. Default values were supplied for the additional information.

The materialized view rawdata.obsinfo is structured as follows:

CREATE MATERIALIZED VIEW rawdata.obsinfo

TABLESPACE pg\_default

AS

SELECT DISTINCT observation.id,

min(result."time") AS startdate,

max(result."time") AS enddate,

'http://5.189.139.50:8080/SensorThingsServer%2D1.0%5FBRGM/v1.0'::text AS namespace,

'http://inspire.ec.europa.eu/codelist/MediaValue/water'::text AS mediamonitored,

'http://inspire.ec.europa.eu/codelist/MeasurementRegimeValue/continuousDataCollection'::text AS measurementregime,

0::boolean AS mobile,

'http://inspire.ec.europa.eu/codelist/ProcessTypeValue/process'::text AS processtype,

'http://inspire.ec.europa.eu/codelist/ResultNatureValue/primary'::text AS resultnature

FROM rawdata.observation,

rawdata.result

WHERE result.observation = observation.id

GROUP BY observation.id, result.observation;

## FROST Materialized Views

In the following sections, the materialized views created as a replacement for the tables utilized by FROST are described.

### MATERIALIZED VIEW DATASTREAMS

CREATE MATERIALIZED VIEW public."DATASTREAMS"

AS

WITH obs\_unit AS (

SELECT DISTINCT observation\_1.id,

unitofmeasure.unit,

unitofmeasure.identifier AS unitdef

FROM rawdata.observation observation\_1,

rawdata.result,

rawdata.unitofmeasure

WHERE observation\_1.id = result.observation AND result.uom = unitofmeasure.id

)

SELECT observation.id AS "ID",

("substring"(observedproperty.identifier::text, 26) || ' '::text) || featureofinterest.description::text AS "DESCRIPTION",

'http://www.opengis.net/def/observationType/OGC-OM/2.0/OM\_Measurement'::text AS "OBSERVATION\_TYPE",

to\_timestamp(to\_char(observation.datestart::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "PHENOMENON\_TIME\_START",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "PHENOMENON\_TIME\_END",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "RESULT\_TIME\_START",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "RESULT\_TIME\_END",

observation.procedure AS "SENSOR\_ID",

observation.observedproperty AS "OBS\_PROPERTY\_ID",

observation.id AS "THING\_ID",

obs\_unit.unit AS "UNIT\_NAME",

obs\_unit.unit AS "UNIT\_SYMBOL",

obs\_unit.unitdef AS "UNIT\_DEFINITION",

("substring"(observedproperty.identifier::text, 26) || ' '::text) || featureofinterest.description::text AS "NAME",

featureofinterest.geom AS "OBSERVED\_AREA",

NULL::text AS "PROPERTIES"

FROM rawdata.observation,

rawdata.observedproperty,

obs\_unit,

rawdata.featureofinterest

WHERE observation.id = obs\_unit.id AND observation.observedproperty = observedproperty.id AND observation.samplingfeature = featureofinterest.id;

ALTER TABLE public."DATASTREAMS"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_datastream

ON public."DATASTREAMS" USING btree

("ID");

### MATERIALIZED VIEW FEATURES

CREATE MATERIALIZED VIEW public."FEATURES"

AS

SELECT foi.id AS "ID",

foi.description AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

st\_asgeojson(foi.geom) AS "FEATURE",

foi.geom AS "GEOM",

foi.identifier::text AS "NAME",

CASE

WHEN foi.sampledfeature IS NULL THEN '{"sampledFeature": "https://sweet.jpl.nasa.gov/2.3/realmHydro.owl#Aquifer"}'::text

ELSE

CASE

WHEN foi2.localidentifier IS NULL THEN '{"sampledFeature": "https://sweet.jpl.nasa.gov/2.3/realmHydro.owl#Aquifer"}'::text

ELSE ('{"sampledFeature": "'::text || foi2.identifier::text) || '"}'::text

END

END AS "PROPERTIES"

FROM rawdata.foi2 foi

LEFT JOIN rawdata.foi2 foi2 ON foi.sampledfeature = foi2.id;

ALTER TABLE public."FEATURES"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_foi

ON public."FEATURES" USING btree

("ID");

### MATERIALIZED VIEW OBSERVATIONS

CREATE MATERIALIZED VIEW public."OBSERVATIONS"

AS

SELECT result.id AS "ID",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "PHENOMENON\_TIME\_START",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "PHENOMENON\_TIME\_END",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "RESULT\_TIME",

result.value AS "RESULT\_NUMBER",

result.value::text AS "RESULT\_STRING",

('{"quality": "'::text || quality.name::text) || '"}'::text AS "RESULT\_QUALITY",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "VALID\_TIME\_START",

to\_timestamp('9999-01-01 01:00:00'::text, 'YYYY-MM-DD HH:MI:SS'::text) AS "VALID\_TIME\_END",

NULL::text AS "PARAMETERS",

observation.id AS "DATASTREAM\_ID",

observation.samplingfeature AS "FEATURE\_ID",

0::smallint AS "RESULT\_TYPE",

NULL::text AS "RESULT\_JSON",

NULL::boolean AS "RESULT\_BOOLEAN",

NULL::bigint AS "MULTI\_DATASTREAM\_ID"

FROM rawdata.result,

rawdata.observation,

rawdata.quality

WHERE result.observation = observation.id AND result.quality = quality.id;

ALTER TABLE public."OBSERVATIONS"

OWNER TO sensorthings;

CREATE INDEX mv\_pk\_obs

ON public."OBSERVATIONS" USING btree

("ID");

### MATERIALIZED VIEW OBS\_PROPERTIES

CREATE MATERIALIZED VIEW public."OBS\_PROPERTIES"

AS

SELECT observedproperty.id AS "ID",

"substring"(observedproperty.identifier::text, 26) AS "NAME",

observedproperty.identifier::text AS "DEFINITION",

observedproperty.identifier::text AS "DESCRIPTION",

NULL::text AS "PROPERTIES"

FROM rawdata.observedproperty;

ALTER TABLE public."OBS\_PROPERTIES"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_obsprop

ON public."OBS\_PROPERTIES" USING btree

("ID");

### MATERIALIZED VIEW SENSORS

CREATE MATERIALIZED VIEW public."SENSORS"

AS

SELECT proc\_uri.id\_raw::bigint AS "ID",

proc\_uri.cit\_name AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

proc\_uri.cit\_link AS "METADATA",

proc\_uri.cit\_shortname AS "NAME",

((((((((('{"namespace": "'::text || proc\_uri.namespace::text) || '", "type":"INSPIRE", "responsibleParty":{"organisationName": "'::text) || proc\_uri.rp\_org::text) || '", "language": "'::text) || proc\_uri.rp\_lang::text) || '", "adminUnit": "'::text) || proc\_uri.rp\_country::text) || '", "script": "'::text) || proc\_uri.rp\_script::text) || '"}}'::text AS "PROPERTIES"

FROM rawdata.proc\_uri;

ALTER TABLE public."SENSORS"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_sensor

ON public."SENSORS" USING btree

("ID");

### MATERIALIZED VIEW THINGS

CREATE MATERIALIZED VIEW public."THINGS"

AS

SELECT observation.id AS "ID",

("substring"(observedproperty.identifier::text, 26) || '\_'::text) || featureofinterest.localidentifier::text AS "DESCRIPTION",

((((((((((((((('{"namespace": "'::text || obsinfo.namespace) || '", "mediamonitored": "'::text) || obsinfo.mediamonitored) || '", "measurementregime": "'::text) || obsinfo.measurementregime) || '", "mobile": "'::text) || obsinfo.mobile) || '", "processtype": "'::text) || obsinfo.processtype) || '", "resultnature": "'::text) || obsinfo.resultnature) || '", "startdate": "'::text) || obsinfo.startdate) || '", "enddate": "'::text) || obsinfo.enddate) || '" }'::text AS "PROPERTIES",

("substring"(observedproperty.identifier::text, 26) || '\_'::text) || featureofinterest.localidentifier::text AS "NAME",

obsinfo.startdate

FROM rawdata.observation,

rawdata.featureofinterest,

rawdata.observedproperty,

rawdata.obsinfo

WHERE observation.samplingfeature = featureofinterest.id AND observation.observedproperty = observedproperty.id AND observation.id = obsinfo.id;

ALTER TABLE public."THINGS"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_thing

ON public."THINGS" USING btree

("ID");

### MATERIALIZED VIEW LOCATIONS

CREATE MATERIALIZED VIEW public."LOCATIONS"

AS

SELECT observation.id AS "ID",

'Forage'::text || foi2.localidentifier::text AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

st\_asgeojson(foi2.geom) AS "LOCATION",

foi2.geom AS "GEOM",

'Forage'::text || foi2.localidentifier::text AS "NAME",

observation.id AS "GEN\_FOI\_ID",

NULL::text AS "PROPERTIES"

FROM rawdata.observation,

rawdata.foi2,

rawdata.observedproperty

WHERE observation.samplingfeature = foi2.id AND observation.observedproperty = observedproperty.id;

ALTER TABLE public."LOCATIONS"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_location

ON public."LOCATIONS" USING btree

("ID");

### MATERIALIZED VIEW THINGS\_LOCATIONS

CREATE MATERIALIZED VIEW public."THINGS\_LOCATIONS"

AS

SELECT observation.id AS "THING\_ID",

observation.id AS "LOCATION\_ID"

FROM rawdata.observation,

rawdata.featureofinterest,

rawdata.observedproperty

WHERE observation.samplingfeature = featureofinterest.id AND observation.observedproperty = observedproperty.id;

ALTER TABLE public."THINGS\_LOCATIONS"

OWNER TO sensorthings;

CREATE INDEX mv\_pk\_locthing

ON public."THINGS\_LOCATIONS" USING btree

("THING\_ID");

CREATE INDEX mv\_pk\_thingloc

ON public."THINGS\_LOCATIONS" USING btree

("LOCATION\_ID");

## FROST Materialized Views URI-ID

While similar to the materialized views described above, the set of materialized views required for the provision of URI-IDs are subtly different. Thus, they are listed in the sections below.

In order to easily tailor the URI-IDs to the relevant configuration, an additional table base\_info has been created, that provides the base URI to be prepended to the baseid in order to form the URI-ID. The base URIs from this table are added to the baseid in the views defined below.

### MATERIALIZED VIEW DATASTREAMS

--DROP MATERIALIZED VIEW public."DATASTREAMS\_V";

CREATE MATERIALIZED VIEW public."DATASTREAMS\_V"

AS

WITH obs\_unit AS (

SELECT DISTINCT observation\_1.baseid,

unitofmeasure.unit,

unitofmeasure.identifier AS unitdef

FROM rawdata.observation observation\_1,

rawdata.result,

rawdata.unitofmeasure

WHERE observation\_1.id = result.observation AND result.uom = unitofmeasure.id

)

SELECT base\_info.uri\_ds::text || observation.baseid AS "ID",

("substring"(observedproperty.identifier::text, 26) || ' '::text) || foi2.description::text AS "DESCRIPTION",

'http://www.opengis.net/def/observationType/OGC-OM/2.0/OM\_Measurement'::text AS "OBSERVATION\_TYPE",

to\_timestamp(to\_char(observation.datestart::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "PHENOMENON\_TIME\_START",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "PHENOMENON\_TIME\_END",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "RESULT\_TIME\_START",

to\_timestamp(to\_char(observation.dateend::timestamp with time zone, 'YYYY-MM-DD'::text), 'YYYY-MM-DD'::text) AS "RESULT\_TIME\_END",

observation.procedure AS "SENSOR\_ID",

observation.observedproperty AS "OBS\_PROPERTY\_ID",

base\_info.uri\_thg::text || observation.baseid::text AS "THING\_ID",

obs\_unit.unit AS "UNIT\_NAME",

obs\_unit.unit AS "UNIT\_SYMBOL",

obs\_unit.unitdef AS "UNIT\_DEFINITION",

("substring"(observedproperty.identifier::text, 26) || ' '::text) || foi2.description::text AS "NAME",

foi2.geom AS "OBSERVED\_AREA",

NULL::text AS "PROPERTIES"

FROM rawdata.observation,

rawdata.observedproperty,

obs\_unit,

rawdata.foi2,

public.base\_info

WHERE observation.baseid::text = obs\_unit.baseid::text AND observation.observedproperty = observedproperty.id AND observation.samplingfeature = foi2.id;

ALTER TABLE public."DATASTREAMS\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_datastream

ON public."DATASTREAMS\_V" USING btree

("ID");

### MATERIALIZED VIEW FEATURES

-- DROP MATERIALIZED VIEW public."FEATURES\_V";

CREATE MATERIALIZED VIEW public."FEATURES\_V"

AS

SELECT base\_info.uri\_foi::text || foi.baseid AS "ID",

foi.description AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

st\_asgeojson(foi.geom) AS "FEATURE",

foi.geom AS "GEOM",

foi.identifier::text AS "NAME",

CASE

WHEN foi.sampledfeature IS NULL THEN '{"sampledFeature": "https://sweet.jpl.nasa.gov/2.3/realmHydro.owl#Aquifer"}'::text

ELSE

CASE

WHEN foi2.localidentifier IS NULL THEN '{"sampledFeature": "https://sweet.jpl.nasa.gov/2.3/realmHydro.owl#Aquifer"}'::text

ELSE ('{"sampledFeature": "'::text || base\_info.uri\_foi::text || foi2.baseid) || '"}'::text

END

END AS "PROPERTIES"

FROM base\_info, rawdata.foi2 foi

LEFT JOIN rawdata.foi2 foi2 ON foi.sampledfeature = foi2.id;

ALTER TABLE public."FEATURES\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_foi\_v

ON public."FEATURES\_V" USING btree

("ID");

### MATERIALIZED VIEW OBSERVATIONS

-- DROP MATERIALIZED VIEW public."OBSERVATIONS\_V";

CREATE MATERIALIZED VIEW public."OBSERVATIONS\_V"

AS

SELECT (base\_info.uri\_res::text || observation.baseid::text || '-'::text) || result.id AS "ID",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "PHENOMENON\_TIME\_START",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "PHENOMENON\_TIME\_END",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "RESULT\_TIME",

result.value AS "RESULT\_NUMBER",

result.value::text AS "RESULT\_STRING",

('{"quality": "'::text || quality.name::text) || '"}'::text AS "RESULT\_QUALITY",

to\_timestamp(to\_char(result."time"::timestamp with time zone, 'YYYY-MM-DD HH:MI:SS'::text), 'YYYY-MM-DD HH:MI:SS'::text) AS "VALID\_TIME\_START",

to\_timestamp('9999-01-01 01:00:00'::text, 'YYYY-MM-DD HH:MI:SS'::text) AS "VALID\_TIME\_END",

NULL::text AS "PARAMETERS",

base\_info.uri\_ds::text || observation.baseid::text AS "DATASTREAM\_ID",

base\_info.uri\_foi::text || 'foi/' || observation.baseid::text AS "FEATURE\_ID",

0::smallint AS "RESULT\_TYPE",

NULL::text AS "RESULT\_JSON",

NULL::boolean AS "RESULT\_BOOLEAN",

NULL::bigint AS "MULTI\_DATASTREAM\_ID"

FROM rawdata.result,

rawdata.observation,

rawdata.quality,

base\_info

WHERE result.observation = observation.id AND result.quality = quality.id;

ALTER TABLE public."OBSERVATIONS\_V"

OWNER TO sensorthings;

CREATE INDEX mv\_pk\_obs

ON public."OBSERVATIONS\_V" USING btree

("ID");

### MATERIALIZED VIEW OBS\_PROPERTIES

-- DROP MATERIALIZED VIEW public."OBS\_PROPERTIES\_V";

CREATE MATERIALIZED VIEW public."OBS\_PROPERTIES\_V"

AS

SELECT observedproperty.id AS "ID",

"substring"(observedproperty.identifier::text, 26) AS "NAME",

observedproperty.identifier::text AS "DEFINITION",

observedproperty.identifier::text AS "DESCRIPTION",

NULL::text AS "PROPERTIES"

FROM rawdata.observedproperty;

ALTER TABLE public."OBS\_PROPERTIES\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_obsprop

ON public."OBS\_PROPERTIES\_V" USING btree

("ID");

### MATERIALIZED VIEW SENSORS

-- DROP MATERIALIZED VIEW public."SENSORS\_V";

CREATE MATERIALIZED VIEW public."SENSORS\_V"

AS

SELECT proc\_uri.id AS "ID",

proc\_uri.cit\_name AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

proc\_uri.cit\_link AS "METADATA",

proc\_uri.cit\_shortname AS "NAME",

((((((((('{"namespace": "'::text || proc\_uri.namespace::text) || '", "type":"INSPIRE", "responsibleParty":{"organisationName": "'::text) || proc\_uri.rp\_org::text) || '", "language": "'::text) || proc\_uri.rp\_lang::text) || '", "adminUnit": "'::text) || proc\_uri.rp\_country::text) || '", "script": "'::text) || proc\_uri.rp\_script::text) || '"}}'::text AS "PROPERTIES"

FROM rawdata.proc\_uri;

ALTER TABLE public."SENSORS\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_sensor

ON public."SENSORS\_V" USING btree

("ID");

### MATERIALIZED VIEW THINGS

--DROP MATERIALIZED VIEW public."THINGS\_V";

CREATE MATERIALIZED VIEW public."THINGS\_V"

TABLESPACE pg\_default

AS

SELECT base\_info.uri\_thg::text || observation.baseid::text AS "ID",

("substring"(observedproperty.identifier::text, 26) || '\_'::text) || foi2.baseid::text AS "DESCRIPTION",

((((((((((((((('{"namespace": "'::text || base\_info.namespace) || '", "mediamonitored": "'::text) || obsinfo.mediamonitored) || '", "measurementregime": "'::text) || obsinfo.measurementregime) || '", "mobile": "'::text) || obsinfo.mobile) || '", "processtype": "'::text) || obsinfo.processtype) || '", "resultnature": "'::text) || obsinfo.resultnature) || '", "startdate": "'::text) || obsinfo.startdate) || '", "enddate": "'::text) || obsinfo.enddate) || '" }'::text AS "PROPERTIES",

("substring"(observedproperty.identifier::text, 26) || '\_'::text) || foi2.baseid::text AS "NAME"

FROM rawdata.observation,

rawdata.foi2,

rawdata.observedproperty,

rawdata.obsinfo,

public.base\_info

WHERE observation.samplingfeature = foi2.id AND observation.observedproperty = observedproperty.id AND observation.id = obsinfo.id;

ALTER TABLE public."THINGS\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_thing

ON public."THINGS\_V" USING btree

("ID");

### MATERIALIZED VIEW LOCATIONS

--DROP MATERIALIZED VIEW public."LOCATIONS\_V";

CREATE MATERIALIZED VIEW public."LOCATIONS\_V"

AS

SELECT base\_info.uri\_loc::text || observation.baseid::text AS "ID",

'Forage'::text || foi2.localidentifier::text AS "DESCRIPTION",

'application/vnd.geo+json'::text AS "ENCODING\_TYPE",

st\_asgeojson(foi2.geom) AS "LOCATION",

foi2.geom AS "GEOM",

'Forage'::text || foi2.localidentifier::text AS "NAME",

base\_info.uri\_foi::text ||'foi/'::text || observation.baseid::text AS "GEN\_FOI\_ID",

NULL::text AS "PROPERTIES"

FROM rawdata.observation,

rawdata.foi2,

rawdata.observedproperty,

base\_info

WHERE observation.samplingfeature = foi2.id AND observation.observedproperty = observedproperty.id

WITH DATA;

ALTER TABLE public."LOCATIONS\_V"

OWNER TO sensorthings;

CREATE UNIQUE INDEX mv\_pk\_location

ON public."LOCATIONS\_V"

("ID");

### MATERIALIZED VIEW THINGS\_LOCATIONS

-- DROP MATERIALIZED VIEW public."THINGS\_LOCATIONS\_V";

CREATE MATERIALIZED VIEW public."THINGS\_LOCATIONS\_V"

AS

SELECT base\_info.uri\_thg::text || observation.baseid::text AS "THING\_ID",

base\_info.uri\_loc::text || observation.baseid::text AS "LOCATION\_ID"

FROM rawdata.observation,

rawdata.featureofinterest,

rawdata.observedproperty,

base\_info

WHERE observation.samplingfeature = featureofinterest.id AND observation.observedproperty = observedproperty.id;

ALTER TABLE public."THINGS\_LOCATIONS\_V"

OWNER TO sensorthings;

CREATE INDEX mv\_pk\_locthing

ON public."THINGS\_LOCATIONS\_V" USING btree

("THING\_ID");

CREATE INDEX mv\_pk\_thingloc

ON public."THINGS\_LOCATIONS\_V" USING btree

("LOCATION\_ID");

## FROST Tables URI-ID

All the database descriptions above have pertained to the utilization of views for provision of measurement data via FROST. Alternatively, utilization of tables has the advantage that FROST becomes truly interactive, allowing for provision of additional data via the SensorThings API. However, when running on tables, FROST automatically generates the identifiers for the individual entries; while this is advantageous when utilizing numeric IDs, it causes difficulties when the data provider wishes to specify a URI as an identifier for an object. In order to allow for provision of identifiers together with the data provided, a trigger function id\_from\_props() has been created, that allows for the following options:

* If an identifier is provided in the ID field, this identifier is utilized (this is the case when data is inserted directly into the DB)
* If an identifier is provided in the JSON Properties section with the name “Id”, this identifier is utilized (this is the case when data is provided via the SensorThings API)
* If no identifier is provided, a UUID is generated and utilized

CREATE FUNCTION public.id\_from\_props()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$

BEGIN

IF NEW."ID" IS NULL THEN

IF (NEW."PROPERTIES"::jsonb)->>'Id' IS NULL THEN

NEW."ID" := uuid\_generate\_v4()::text;

ELSE

NEW."ID" := (NEW."PROPERTIES"::jsonb)->>'Id'::text;

END IF;

END IF;

RETURN NEW;

END

$BODY$;

ALTER FUNCTION public.id\_from\_props()

OWNER TO sensorthings;

As SensorThings Observations already provide a PARAMETERS field, the PROPERTIES extension is not required for this class. Thus, the trigger function id\_from\_param() has also been created, providing the same functionality as id\_from\_props(), but taking the ID from the PARAMETERS entry as follows

CREATE FUNCTION public.id\_from\_param()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$

BEGIN

IF NEW."ID" IS NULL THEN

IF (NEW."PARAMETERS"::jsonb)->>'Id' IS NULL THEN

NEW."ID" := uuid\_generate\_v4()::text;

ELSE

NEW."ID" := (NEW."PARAMETERS"::jsonb)->>'Id'::text;

END IF;

END IF;

RETURN NEW;

END

$BODY$;

ALTER FUNCTION public.id\_from\_param()

OWNER TO sensorthings;