OGC SensorThings API
and
FROST Server®

Michael Jacoby

Fraunhofer IOSB
Open Geospatial Consortium

http://www.opengeospatial.org

- International consortium
  - over 522 companies, government agencies and universities
- “Geo-enable” mainstream IT
- Develop publicly available standards
  - Web Map Service
  - CityGML
  - WaterML
  - Earth Observations
OGC & IoT?

- IoT deals with Sensors and Actuators
- Sensors and Actuators have Location
- OGC Sensor Web Enablement (SWE)
  - Enable developers to make *all types* of sensors and sensor data repositories discoverable, accessible and useable via the Web
  - Since 1990 by NASA
  - Since 2001 in OGC
- SensorML
- Sensor Observation Service (SOS)
- Sensor Planning Service (SPS)
- Observations & Measurements (O&M)
- Sensor Data & Metadata

©OGC: http://www.opengeospatial.org/ogc/markets-technologies/swe
From SWE to SensorThings

- "Old" SWE Standards
  - XML Encoded
  - SOAP bindings
  - Complex in use
    - No easy browsing
    - No pagination
    - No pub/sub

Time for an update → SensorThings API
OGC SensorThings API

https://www.opengeospatial.org/standards/sensorthings
https://github.com/opengeospatial/sensorthings

- A standard for exchanging sensor data and metadata
  - Historic data & current data
  - JSON Encoded
  - RESTful
  - Adapting OASIS OData URL patterns and query options
  - Supporting MQTT pub/sub

- Easy to use & understand
  - Discoverable with only a web browser
OGC SensorThings API

- Divided into multiple Parts
  - Part I: Sensing (published 07/2016)
  - Part II: Tasking Core (published 01/2019)
  - Part III: Rule Engine

Part I: Sensing

- Mandatory: Basic read access
- Extensions
  - Filtering
  - Create/Update/Delete
  - Batch Processing
  - MultiDatastreams
  - Data Arrays
  - MQTT: Create Observations
  - MQTT: Receive updates

- Conformance Test Suite
How does it work?

- Data Model
  - What kind of entities exist?
  - How are they connected?

- API
  - Basic read access
  - Filtering
  - Create/Update/Delete
Data Model
HTTP API: Basic operations

- **Base URL:** `http://server.org/FROST-Server/v1.0`

- **Read:** HTTP GET
  - `v1.0` → Get collection index
  - `v1.0/Collection` → Get all entities in a collection
  - `v1.0/Collection(id)` → Get one entity from a collection

- **Create:** HTTP POST
  - `v1.0/Collection` → Create a new entity

- **Update:** HTTP PATCH
  - `v1.0/Collection(id)` → Update an entity

- **Update:** HTTP PUT
  - `v1.0/Collection(id)` → Replace an entity

- **Delete:** HTTP DELETE
  - `v1.0/Collection(id)` → Remove an entity
HTTP API: Get a Collection

HTTP GET v1.0/Things

Response

```json
{
   "value": [
      {
         "name": "My camping lantern",
         "description": "camping lantern",
         "properties": {
            "property1": "it’s waterproof",
            "property2": "it glows in the dark"
         },
         "Locations@iot.navigationLink": "Things(1)/Locations",
         "HistoricalLocations@iot.navigationLink": "Things(1)/HistoricalLocations",
         "Datastreams@iot.navigationLink": "Things(1)/Datastreams",
         "@iot.id": 1,
         "@iot.selfLink": "/FROST-Server/v1.0/Things(1)"
      },
      {
         a second thing...
      },
      { ... },
      { ... },
      { ... }
   ]
}
```
HTTP API: Get an Entity

HTTP GET v1.0/Things(1)

- Response

```json
{
    "name" : "My camping lantern",
    "description" : "camping lantern",
    "properties" : {
        "property1" : "it’s waterproof",
        "property2" : "it glows in the dark"
    },
    "Locations@iot.navigationLink" : "Things(1)/Locations",
    "HistoricalLocations@iot.navigationLink" : "Things(1)/HistoricalLocations",
    "Datastreams@iot.navigationLink" : "Things(1)/Datastreams",
    "@iot.id" : 1,
    "@iot.selfLink" : "/FROST-Server/v1.0/Things(1)"
}
```
HTTP API: Get related Entities

Get all Datastreams of a specific Thing

- **HTTP GET** v1.0/Things(1)/Datastreams

- **Response**

  
  ```json
  {
    "value" : [
      {...},
      {...},
      {...},
      {...}
    ]
  }
  ```
HTTP API: Pagination

Get only 4 Observations and the total count of Observations

- **HTTP GET** v1.0/Observations?$top=4&$count=true

- **Response**

```json
{
    "@iot.count" : 16,
    "@iot.nextLink" : "/FROST-Server/v1.0/Observations?$top=4&$skip=4",
    "value" : [
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```
HTTP API: Sorting

Get all Observations sorted by phenomenonTime, newest first

- HTTP GET `v1.0/Observations?$orderby=phenomenonTime desc`

- Functions work for Ordering

  HTTP GET `v1.0/Datastreams?$orderby=length(name) desc`
HTTP API: Filtering

Get only Observations with result (value) > 5

- **HTTP GET** `v1.0/Observations?$filter=result gt 5`

- **Response**

  ```json
  {
    "@iot.nextLink" : "/FROST-Server/v1.0/Observations?$filter=result gt 5&$top=4&$skip=4",
    "value" : [ 
        {
          "resultTime" : null,
          "result" : 10,
          "@iot.id" : 34,
          "@iot.selfLink" : "/FROST-Server/v1.0/Observations(34)"
        },
        { ... 
        },
        { ... 
        },
        { ... 
        }
    ]
  }
  ```
HTTP API: Filtering Functions 1

- **Comparison Operators**
  - `gt`
  - `ge`
  - `eq`
  - `le`
  - `lt`
  - `ne`

- **Logical Operators**
  - `and`
  - `or`
  - `not`

- **Mathematical Operators**
  - `add`
  - `sub`
  - `mul`
  - `div`
  - `mod`

- **String Functions**
  - `substringof(p0, p1)`
  - `endswith(p0, p1)`
  - `startswith(p0, p1)`
  - `substring(p0, p1)`
  - `indexof(p0, p1)`
  - `length(p0)`
  - `tolower(p0)`
  - `toupper(p0)`
  - `trim(p0)`
  - `concat(p0, p1)`

- **Mathematical Functions**
  - `round(n1)`
  - `floor(n1)`
  - `ceiling(n1)`
HTTP API: Filtering Functions 2

- **Geospatial Functions**
  - geo.intersects(g1, g2)
  - geo.length(l1)
  - geo.distance(g1, g2)
  - st_equals(g1, g2)
  - st_disjoint(g1, g2)
  - st_touches(g1, g2)
  - st_within(g1, g2)
  - st_overlaps(g1, g2)
  - st_crosses(g1, g2)
  - st_intersects(g1, g2)
  - st_contains(g1, g2)
  - st_relate(g1, g2)

- **Date and Time Functions**
  - now()
  - mindatetime()
  - maxdatetime()
  - date(t1)
  - time(t1)
  - year(t1)
  - month(t1)
  - day(t1)
  - hour(t1)
  - minute(t1)
  - second(t1)
  - fractionalseconds(t1)
  - totaloffsetminutes(t1)
HTTP API: Filtering examples

- All observations with an even result
  - v1.0/Observations?$filter=result mod 2 eq 0

- Observations of the last hour
  - v1.0/Observations?$filter=phenomenonTime gt now() sub duration'PT1H'

- Datastreams that measure temperature
  - v1.0/Datastreams?$filter=ObservedProperty/name eq 'temperature'
HTTP API: $select

Get only description und id for all Things

- **HTTP GET** `v1.0/Things?$select=@iot.id,description`

- **Response**

```json
{
    "value" : [
        {
            "description" : "camping lantern",
            "@iot.id" : 1
        },
        {
            "description" : "camping stove",
            "@iot.id" : 2
        }
    ]
}
```
HTTP API: $expand

Get the Thing with id=17 and its Datastreams

- **HTTP GET** v1.0/Things(17)?$expand=Datastreams

- **Response**

```json
{
    "name" : "My camping lantern",
    "description" : "camping lantern",
    "Datastreams" : [
        { ... },
        { ... },
        { ... }
    ],
    "@iot.id" : 17
}
```
HTTP API: $expand( ... )

Get only description, id and Datastreams for Thing 17 and for the Datastreams only id and description:

- **HTTP GET** `v1.0/Things(17)?$select=@iot.id,description&$expand=Datastreams($select=@iot.id,description)`

- **Response**

```json
{
  "description" : "camping lantern",
  "@iot.id" : 17,
  "Datastreams" : [
    {
      "description" : "Temperature measurement",
      "@iot.id" : 19
    },
    {
      "description" : "Humidity measurement",
      "@iot.id" : 21
    }
  ]
}
```
Questions?
FROST Server®

Michael Jacoby

Fraunhofer IOSB
**FROST® Server**

https://github.com/FraunhoferIOSB/FROST-Server (Code)
http://akme-a3.iosb.fraunhofer.de/FROST-Server/v1.0 (Demo)

- **FRaunhofer Open Source SensorThings API Server**
  - LGPL 3.0 license
  - First implementation to include all extension
    - Based on JavaEE / PostgreSQL / PostGIS
  - Official OGC reference implementation
- **High scalability**
  - Single-board computers (e.g. RaspberryPi)
  - Local server (clusters)
  - Cloud/Data Center
Agenda

- History
- Features
- Deployment
- FROST Landscape
History

- 2016-02: Start of development
  - Goal: A full implementation of the STA
- 2016-07: Open-Source (LGPL) on GitHub
- 2016-11: v1.0 – CRUD, DataArray, MQTT
- 2016-11: MultiDatastream
- 2017-01: JSON filtering
- 2017-09: Docker support
- 2018-01: StringID & UUID Backends
- 2018-02: Batch Processing
  - Goal reached!
- 2018-04: Horizontal scalability
- 2018-04: Client-specified IDs
- 2018-08: HELM chart
- 2019-07: Tasking
Data Type Handling

- Observation/result has type **Any**
- **Any? Anything that is valid in JSON**
  - Number: 1.23e-3
  - String: "cloudy"
  - Object: {"temp": 1.2, "clouds": true}
  - Array: [1.2, 1.3, 0.9]
  - Boolean: true / false
  - No-Value: null

FROST Server

- Truly type-conserving
- Type-specific ordering
- Type-safe filtering
Properties for all Entity Types

- In STA v1.0 for Things and Observations
  - Thing/properties
  - Observation/parameters
- Great for storing metadata related to external systems
- Properties for other entity types
  - Datastream/properties
  - MultiDatastream/properties
  - FeatureOfInterest/properties
  - Location/properties
  - ObservedProperty/properties
  - Sensor/properties
Filtering on sub-properties of JSON Objects

- **Type-safe filtering**
  - v1.0/Things? $filter=properties/type eq 'room'
  - v1.0/Things? $filter=Locations/properties/floor eq 2
  - v1.0/Things? $filter=properties/enabled

- **Array access**
  - v1.0/Things? $filter=properties/sizes[2][0]/length gt 1.1e-6
Filtering for time intervals

- `<`, `<=`, `==`, `>=`, `>` are not enough when comparing time intervals
- Allen’s Interval Algebra
  - before / after
  - meets
  - overlaps
  - starts
  - during
  - finishes

Example

$v1.0/MultiDatastreams(1)/Observations?
$filter=overlaps(phenomenonTime,2018-01-01T00:00:00Z/P1D)$
Filtering Delete

- STA v1.0 only allows delete on single entities
- DELETE on Collections with $filter support
  - DELETE v1.0/Observations?
    $filter=phenomenonTime gt now() sub duration'P1M'
  - DELETE v1.0/Datastream(1)/Observations?
    $filter=phenomenonTime gt now() sub duration'P1D' mul Datastreams/Sensor/properties/keepDays
ID Handling

- Supported ID types in FROST-Server
  - Long (default) {"@iot.id": 12345}
  - UUID {"@iot.id": "123e4567-e89b-12d3-a456-426655440000"}
  - String {"@iot.id": "http://example.org/ontology/superThing"}

- ID generation methods
  - Server defined (default)
  - User defined
  - Mixed
Deployment

- Two options
  - All-In-One
  - Separated HTTP and MQTT

- Deploy as
  - Docker
    - Docker images
      [https://hub.docker.com/u/fraunhoferiosb/](https://hub.docker.com/u/fraunhoferiosb/)
    - Docker-compose examples
      [https://github.com/FraunhoferIOSB/FROST-Server](https://github.com/FraunhoferIOSB/FROST-Server)
    - Helm chart (for deployment on Kubernetes)
      [https://github.com/FraunhoferIOSB/helm-charts](https://github.com/FraunhoferIOSB/helm-charts)
  - Tomcat
    - Spring Boot Application (via Kinota™ Server)
      [https://github.com/kinota/kinota-server](https://github.com/kinota/kinota-server)
Deployment: All-In-One

- Database
  - Store Entity
  - Check Subscription

- JVM (Tomcat)
  - Entity Changed

- HTTP
  - 1: Patch /Datastreams(1)

- MQTT
  - /Datastreams
  - /Datastreams(1)
  - /Thing(1)/Datastreams
  - /Sensor(1)/Datastreams
  - /ObservedProperties(1)/Datastreams
  - /Observation(...)/DataStream
Deployment: Separated HTTP AND MQTT

Database

Message Bus

Entity Changed

JVM (Tomcat)  JVM (Tomcat)  JVM  JVM

HTTP  HTTP  MQTT  MQTT

1: Patch /Datastreams(1)

/Datastreams
/Datastreams(1)
/Thing(1)/Datastreams

...
Deployment: Docker

- Install docker & docker-compose
- Download docker-compose file
  - All-In-One
    https://raw.githubusercontent.com/FraunhoferIOSB/FROST-Server/master/docker-compose.yaml
  - Separated HTTP and MQTT
    https://raw.githubusercontent.com/FraunhoferIOSB/FROST-Server/master/docker-compose-separated.yaml
- Run docker-compose
  - All-In-One
    > docker-compose up
  - Separated HTTP and MQTT
    > docker-compose -f docker-compose-separated.yaml up
- Open browser on http://localhost:8080/FROST-Server/v1.0
FROST Landscape

- Officially released https://github.com/FraunhoferIOSB/...
  - FROST-Server
  - FROST-Client
  - FROST-Manager
  - helm-charts

- Experimental Tools https://github.com/hylkevds/...
  - SensorThings-Dashboard
  - SensorThingsProcessor
  - SensorThingsImporter
  - SensorThingsCopier
Questions?

- OGC SensorThings API
  https://www.opengeospatial.org/standards/sensorthings
  https://github.com/opengeospatial/sensorthings

- FROST® Server
  https://github.com/FraunhoferIOSB/FROST-Server

- michael.jacoby@iosb.fraunhofer.de

Install FROST Server in only 3 simple steps NOW!

- `wget https://github.com/FraunhoferIOSB/FROST-Server/blob/master/docker-compose.yaml`
- `docker-compose up`
- `open http://localhost:8080/FROST-Server/v1.0`