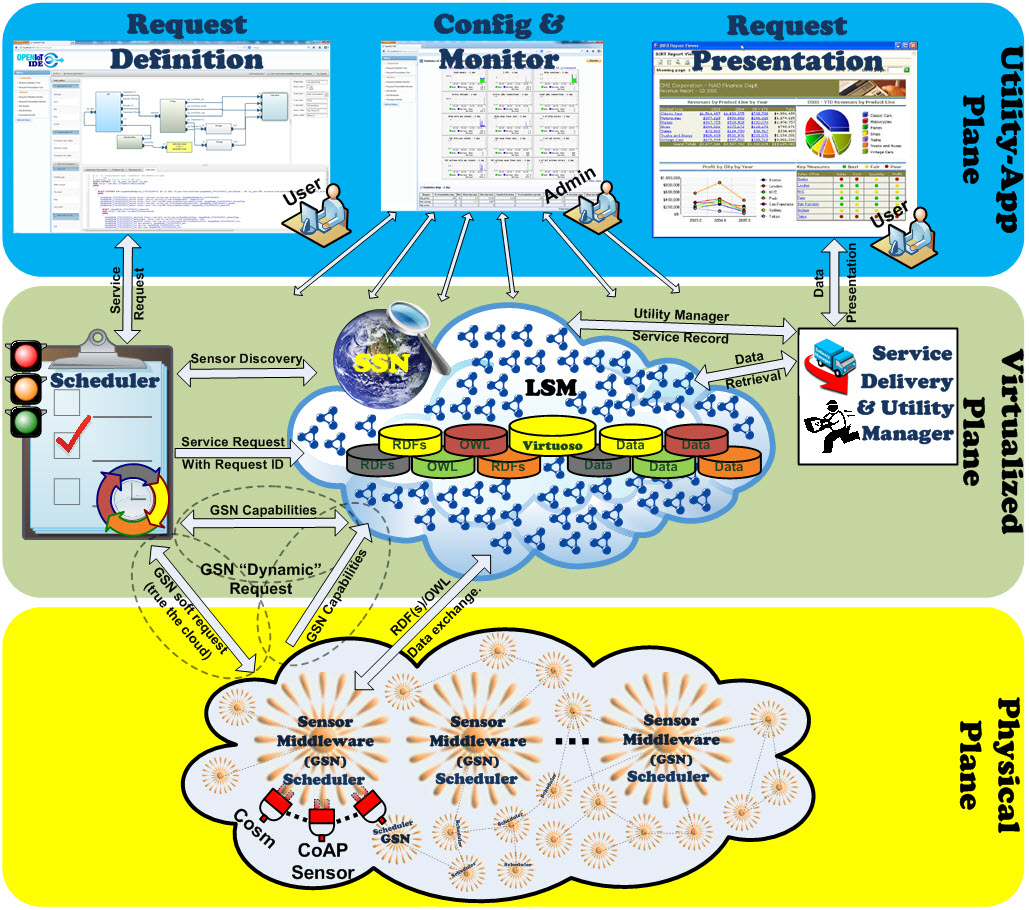
**OpenIoT**

The OpenIoT is an open source middleware platform for implementing and integrating the Internet-of-Things(IoT) solutions. OpenIoT allows the user to easily integrate and use any type of sensors to the OpenIoT platform. OpenIoT already supports several devices, but developers can implement / support any additional devices with only a reasonable effort.

The OpenIoT provides various features such as:

* Collecting and processing data from any sensors (both physical and virtual)
* Semantic Interoperability by annotating sensor data using W3C standards (SSN)
* Streaming sensor data to cloud services.
* Dynamic discovery, query and integration of sensor data.
* Deploying integrated services by comprising multiple sensors data
* Visualization of IoT data for mashup generation such as charts, graphs and maps and data analytics

The architecture of the OpenIoT is shown in Figure.

*[Figure No] OpenIoT architecture [Reference No]*

The OpenIoT platform consists of three planes, Utility/Application Plane, Virtualized Plane and Physical Plane. The utility/Application plane has 3 components for the user.

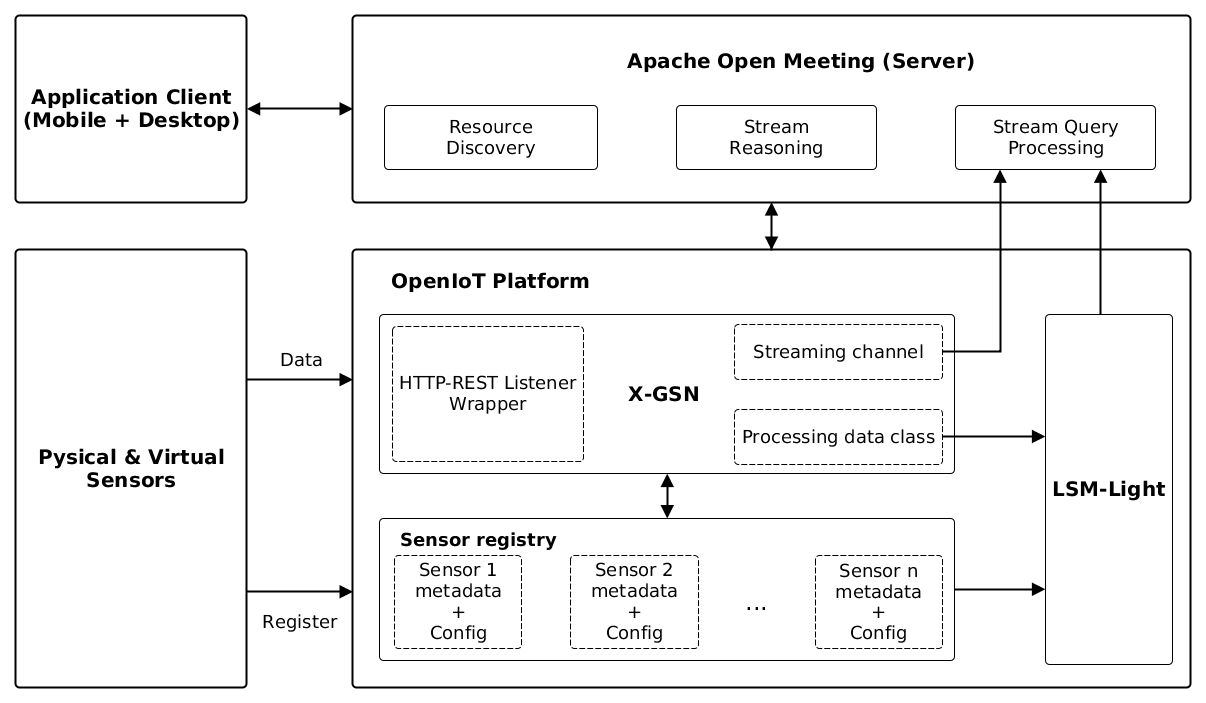
The Request Definition and the Request Presentation components are web interfaces to discover sensors, mashup and visualise the data. The Configuration and Monitoring component enables the user to monitor the health of the different deployed modules.

The Virtualized Plane consist of three components, Scheduler, Cloud Data Storage and Service Delivery & Utility Manager.

The physical plane is the sensor middleware and is the interface to the physical world.

[Reference No?]

The OpenIoT platform is used as a component of the IoE application architecture, particularly *X-GSN* for collecting sensor data from any sensors and *LSM-Light* for storing RDF triples shown in Figure.

*[Figure No] Overall system architecture*

**X-GSN**

X-GSN is the sensor middleware extended Global Sensor Network (GSN). X-GSN collects, filters and annotates data stream semantically from virtual and physical sensors devices. [reference No ?]. Sensors need to be registered to the OpenIoT platform.

## **Configuring GSN for integration with LSM**

LSM schema and URL of LSM-Light server to use have to be specified in *lsm\_config.properties* in orderto connect X-GSN with LSM-Light.

The metaGraph is used to store OpenIoT sensor metadata and the dataGraph is used to store OpenIoT sensor data.

*[Figure No] lsm\_config.properties *

**X-GSN Wrappers:**

X-GSN supports various wrappers such as a CSV wrapper which loads data from a CSV file, HTTP-GET wrapper which handles HTTP GET requests and so on. Each wrapper is a thread in GSN. Also custom wrappers can be created. The wrapper names and classes have to be defined in *wrappers.properties*. The HTTP-REST Listener wrapper created for IoE application is used to receive raw sensor data from virtual/physical sensors. More details will be described in the later section.

**Processing Class:**

A processing class in X-GSN handles processing data between the wrapper and the data publishing engine. Various processing classes are supported and also custom processing classes can be created.

The LSMExporter processing class provided by default by X-GSN is used to push data to the LSM-Light server.

**LSM**

The Linked Stream Middleware Light (LSM-Light) is the cloud data storage. The data streams from X-GSN are transformed into a Linked Data representation, e.g. RDF triples and stores them into Virtuoso Universal Server.

The LSM-Light provides pull-based and push-based approaches to import/stream data to the LSM-Light. In pull-based approach, the LSM-Light pulls periodically data streams. In push-based approach, sensor data is sent to the LSM-Light.[reference No]

In IoE application, push-based approach is used.

**Sensor registration**

Each sensor has to be registered to the LSM-Light. In order to register, it is necessary to create a sensor metadata file and a sensor configuration XML file. Both file names have to be the same.

**Creating sensor metadata file**

The metadata file contains metadata about the sensors. An example of a sensor metadata file is shown in Figure.

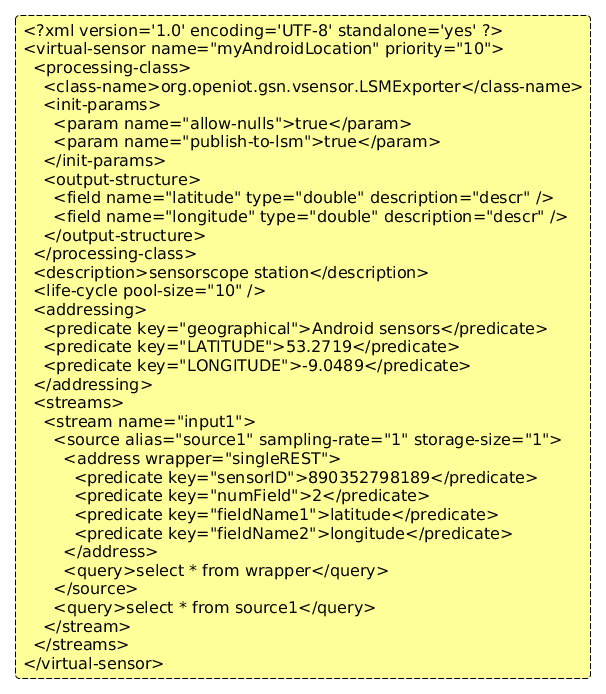
*[Figure No] sensor metadata file*

*[Table No] Description of some fields*

|  |  |
| --- | --- |
| Field type | Description |
| sensorName | Define name of sensor, it should be an unique name |
| metaGraph | Define LSM schema to use specified in lsm\_config.properties |
| dataGraph | Define LSM schema to use specified in lsm\_config.properties |
| fields | Define observation types, they can be defined in multiple comma separated fields |
| field.FIELD\_NAME.  propertyName | Define URI of the FIELD\_NAME property |
| field.FIELD\_NAME.unit | Define observation unit |
| latitude/longitude | Define a sensor location coordinates |
| sensorID | Initially,it should be empty. When registering a sensor, a unique sensorID will be provided by the LSM-Light, then replace it |

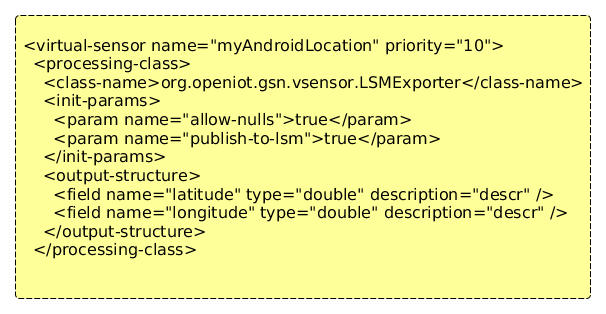
**Creating configuration XML**

A wrapper class and a processing class have to be specified in this configuration file. An example of a sensor configuration XML file is shown in Figure.

*[Figure No] Configuration XML file*

**Specify processing class**

The processing class requires the following parameters to process data to data publishing engine as shown in Figure.

*[Figure No] Specify processing class*

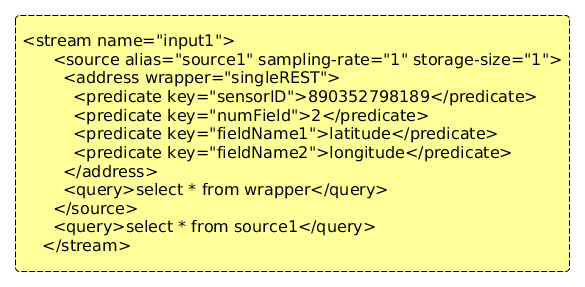
*[Table No] Description of parameters*

|  |  |
| --- | --- |
| Parameters | Description |
| <virtual-sensor name> | Name of the sensor which is defined in the metadata file |
| <class-name> | Define a processing class |
| <field name, type> | Define field name as field which is defined in the metadata file  Define data types. GSN supports TINYINT, SMALLINT, INTEGER, BIGINT, CHAR(#), BINARY[(#)], VARCHAR(#), DOUBLE, TIME |

**Specify wrapper class**

When X-GSN starts up, the wrapper parameters will be initialised.

Each wrapper has different wrapper parameters and the HTTP-REST Listener wrapper uses the following parameters as shown in Figure.



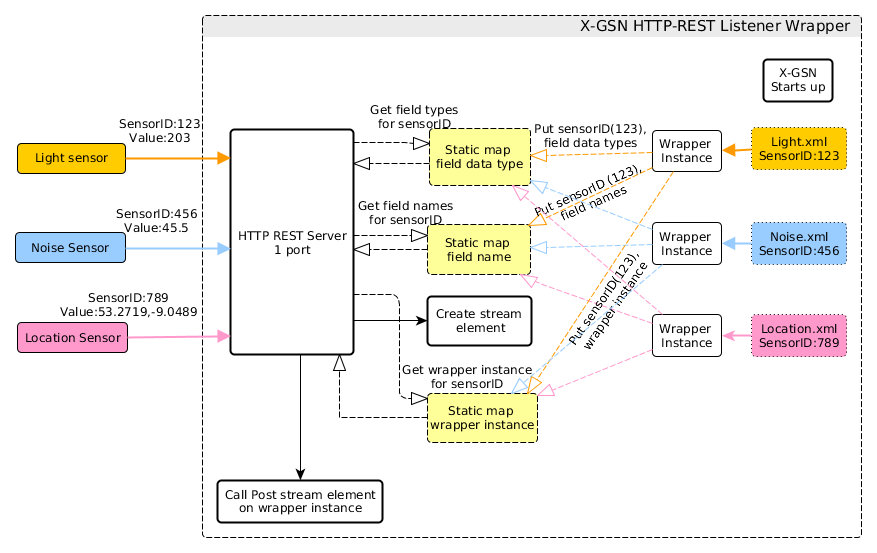
*[Figure No] Specify a wrapper*

*[Table No] Description of parameters*

|  |  |
| --- | --- |
| Parameters | Description |
| <address wrapper> | Name of a wrapper defined in wrappers.properties |
| <predicate key="sensorID"> | Define the sensorID |
| <predicate key="numField"> | Define the number of fields, required if more than one field |
| <predicate key="fieldName*N*"> | Define the field names  If the fields are more than one, define fieldName2, fieldName3 ... fieldName*N* etc. |

The script, *lsm-register.sh/lsm-register.bat,* is provided to register sensors to LSM. This script takes an argument of the metadata file. Also sensors can be registered to LSM through an RDF file or a REST service.

**HTTP-REST** **Listener Wrapper:**

****

*[Figure No] HTTP-REST Listener Wrapper*

The HTTP-REST Listener wrapper handles receiving raw sensor data from virtual/physical sensor devices and generating stream elements and sending the streamed data to registered stream processors. The architecture of the HTTP-REST Listener wrapper is shown in Figure.

The REST service is initialised once and a HTTP server connector is added and starts running when X-GSN launches. One port is opened and listening to clients. The HTTP-REST Wrapper handles GET and POST requests.

Each sensor has its sensor ID which is registered to LSM-Light sever and sends sensor raw data with the sensorID.

When X-GSN starts up, the wrapper parameters which are defined in the sensor configuration XML are loaded and initialised. The wrapper parameter field names, data field types will be stored in maps with the sensorID as the key.

When the HTTP server receives sensor data, the stream elements are created and sent by the wrapper mapped with the sensorID.

