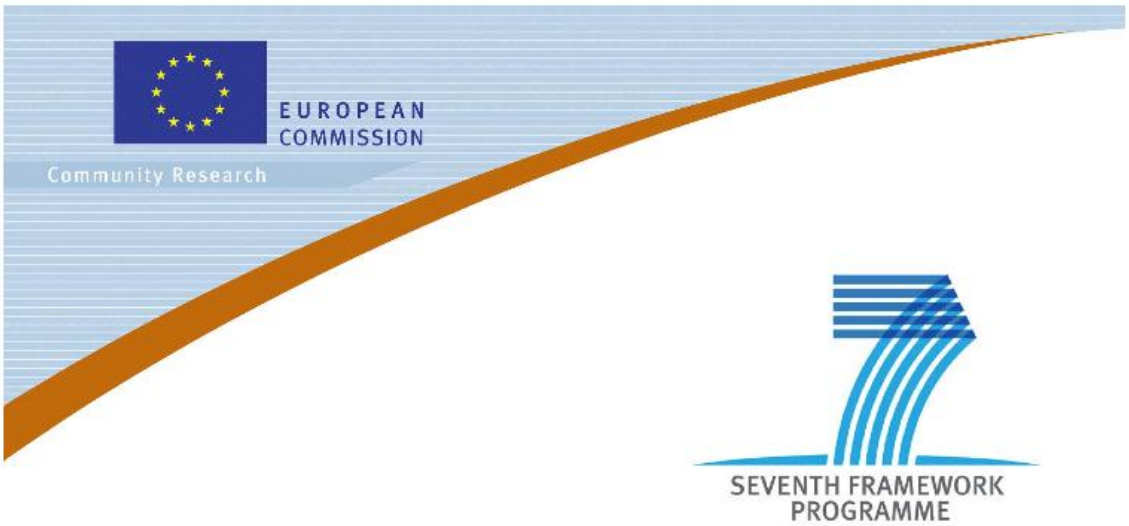


Private Public Partnership Project (PPP)
Large-scale Integrated Project (IP)

D.5.4.1: FI-WARE User and Programmers Guide

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP5-D.5.4.1
Project Document Date: 2012-11-16
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the usage of each Generic Enabler provided by the "Internet of Things Service Enablement chapter.
This document consolidates new contents and also contents in previous issues of Release 1. The reason for re-delivering parts that were already issued is twofold:

- FI-WARE has made an effort to create a unified and improved format. The parts generated in the past are also provided in the new enhanced format for the sake of uniformity and readability.
- A single reference document per chapter is clearer and easier to handle that two incremental issues.

1.2 About This Document

This document comes along with the Software implementation of components, each release of the document being referred to the corresponding Software release (as per D.x.2), to provide documentation of the features offered by the components and interfaces to users/adopters. Moreover, it explains the way they can be exploited in their developments.

1.3 Intended Audience

The document targets users as well as programmers of FI-WARE Generic Enablers.

1.4 Chapter Context

FI-WARE will build the relevant Generic Enablers for Internet of Things Service Enablement, in order for things to become citizens of the Internet – available, searchable, accessible, and usable – and for FI services to create value from real-world interaction enabled by the ubiquity of heterogeneous and resource-constrained devices.
Nota Bene: For the reader, we are using in the following chapters the same vocabulary than in the FI-Ware Product Vision chapter
Thing. Physical object, living organism, person or concept interesting from the perspective of an application.
Device. Hardware entity, component or system that either measures properties of a thing/group of things or influences the properties of a thing/group of things or both measures/influences. Sensors and actuators are devices.
IoT Gateway. A device hosting a number of features of one or several Generic Enablers of the IoT Service Enablement. It is usually located at proximity of the devices to be connected.
IoT Resource. Computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. The resource is usually hosted on the device.
The deployment of the architecture of the IoT Service Enablement chapter is typically distributed across a large number of Devices, several Gateways and the Backend. The Generic Enablers described in this chapter, shown in the figure below, implement functionalities distributed across IoT resources hosted by devices, IoT Gateways and in the IoT Backend.

Device and IoT Resource

A device is a hardware entity, component or system that either measures properties of a thing/group of things or influences the properties of a thing/group of things or both measures/influences. Sensors and actuators are devices. Devices can further be categorized into IoT compliant (i.e., devices with the full-blown FI-WARE capabilities and supporting the standard ETSI M2M interface) and non-compliant (legacy devices with propietary protocols).
IoT Resources are computational elements (software) that provide the technical means to perform sensing and/or actuation on the device. The resource is usually hosted on the device.

Gateway

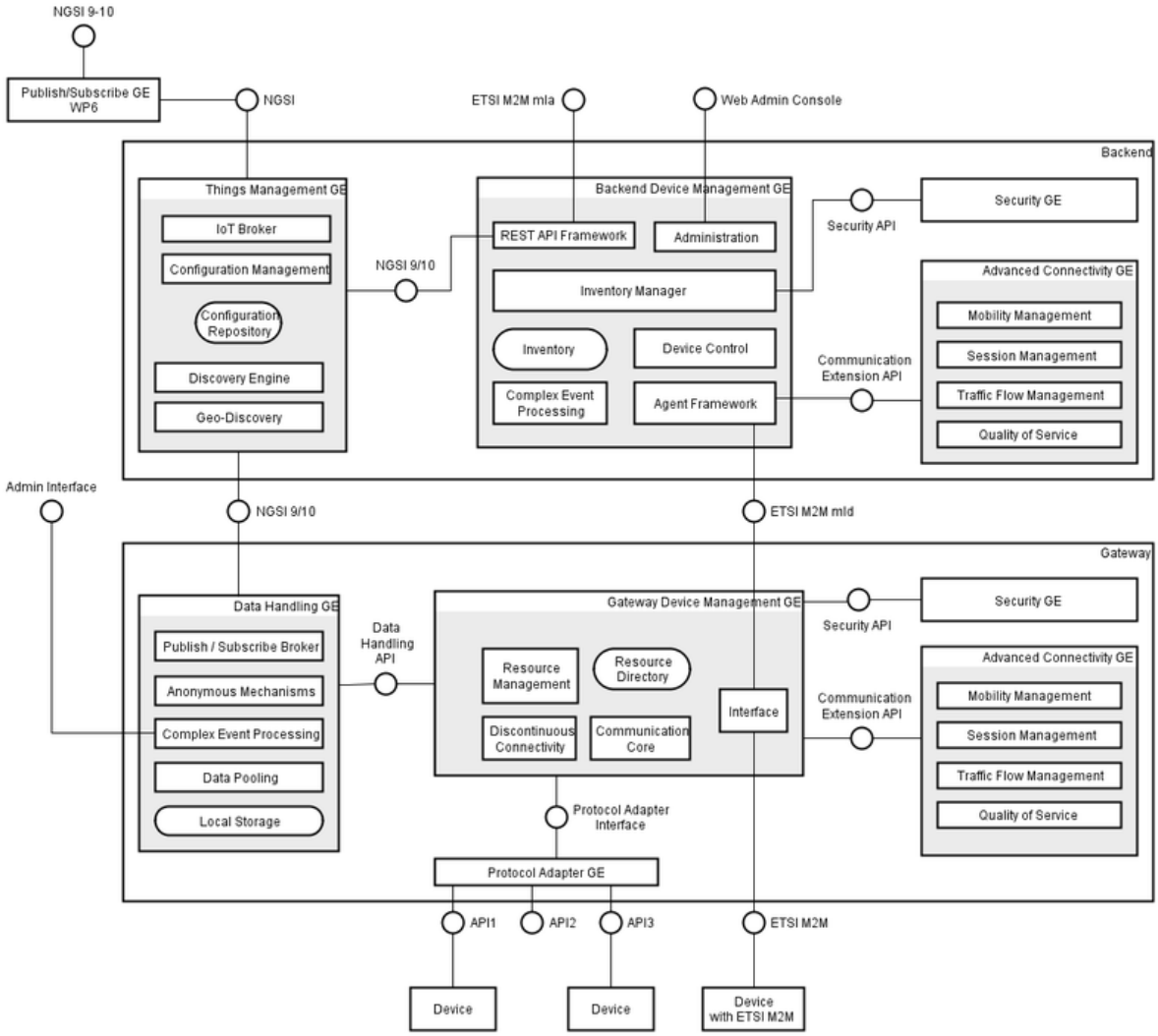
A gateway is providing inter-networking and protocol conversion functionalities between devices and the IoT backend. It is usually located at proximity of the devices to be connected. An example of an IoT gateway is a home gateway that may represent an

aggregation point for all the sensors/actuators inside a smart home. The IoT gateway will support all the IoT backend features, taking into consideration the local constraints of gateway devices such as the available computing, power, storage and energy consumption. Gateways are connected northbound to the backend via IP connectivity and southbound to IoT compliant devices without IP connectivity Legacy devices that needs protocol conversion As IP devices will now appear on the market, the gateway will also be able to manage some of them using IETF Core CoaP protocol but one of the main role of the gateway is to bridge different technologies with IP connectivity. The second main role is deployment of smart services as close as possible of the things to enphazise smart applications development.

Backend

The backend provides management functionalities for the devices and IoT domain-specific support for the applications. It supports access at both IoT resource and thing-level. The backend can be connected southbound to gateways and/or IoT compliant devices (devices that will implement the standardised interface i.e. ETSI M2M).

Current developments are focusing on Backend and Gateway interactions.



More information about the IoT Service Enablement Chapter and FI-WARE in general can be found within the following pages:

http://wiki.fi-ware.eu
Internet_of_Things_Services_Enablement_Architecture
Materializing_Internet-Of-Things-Services-Enablement_in_FI-Ware

1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/

The following resources were used to generate this document:

D.5.4.1b FI-WARE User and Programmers Guide front page
Backend Things Management - Configuration Management User and Programmers Guide
Backend Things Management - IoT Broker User and Programmers Guide
Gateway Device Management - User and Programmers Guide
Gateway Protocol Adapter (ZPA) - User and Programmers Guide
Gateway Data Handling - SOL CEP User and Programmers Guide
Gateway Data Handling - Esper4FastData Mobile - User and Programmers Guide
Gateway Data Handling - Esper4FastData Servlet - User and Programmers Guide

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.
This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.
Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the

rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as the following partners: Atos, Ericsson, NEC, Orange, Telefonica and Telecom Italia .

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor
v0	First draft of deliverable submission generated	2012-11-15	Automated
v1	Deliverable ready for submission	2012-11-16	FT
v2	Deliverable ready for submission	2012-11-16	TID, SAP

1.10 Table of Contents

2 Backend Things Management - Configuration Management User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

2.1 Introduction

The Configuration Management component of the Things Management GE is the component responsible for the registration and discovery of entities. The executable is distributed as an RPM package and installed as /usr/bin/iotConfigMgr. iotConfigMgr uses a MySQL database as persistant storage and it is highly recommended for the MySQL server to run in the same node as the iotConfigMgr.
iotConfigMgr is a REST server implementing the FIWARE NGSI-9 interface, using the name FIWARE and not OMA to indicate that the inbterface used may haqve slight variations from what OMA publishes. In the first FIWARE release, the iotConfigMgr will support only two of the NGSI-9 requests, namely:

- registerContextRequest
- discoverContextRequest

2.2 Starting iotConfigMgr

iotConfigMgr supports a number of command line options, the most important being:

- -u (print the complete usage on screen)
- -fg (run in foreground)
- -port (port to receive new connections)
- -reset (reset database at startup)
- -dbhost (host where the database server runs)
- -dbuser (username to login to database)
- -dbpwd (password to login to database)
- -db (name of the database)
- -psbHost (hostname for Pub/Sub Broker)
- -psbPort (port for Pub/Sub Broker)

The first time iotConfigMgr is started it will find an empty database and will then create all the necessary tables for it to function. A complete reset of the database is accomplished using the '-reset' option at starting iotConfigMgr. The '-fg' option is used when debugging iotConfigMgr, avoiding the executable to turn itself into a daemon process. To change the port where iotConfigMgr accepts incoming connections, the '-port' option is used. Starting iotConfigMgr with the '-u' options makes it print the full usage on the screen and after that it dies.

2.3 Database

Before starting iotConfigMgr, a MySQL server need to be running and the database to be used must be created and configured to that the database user of iotConfigMgr has access to it. All this is explained in the 'Installation and Administration Guide'.

The name of the database is 'cm' by default and this name is changed using the command line option '-db'.
The name of the database server is 'localhost' by default and this name is changed using the command line option '-dbhost'.
The name of the database user is 'cm' by default and this name is changed using the command line option '-dbuser'.
The name of the database password is 'cm' by default and this name is changed using the command line option '-dbpwd'.

The database tables used are the following:

- attribute
- attributeMetadata
- entity
- entityAttribute
- metadata
- registration
- registrationMetadata

To enter mysql and check what's in the database, the following mysql client command is used:

```
% mysql -u cm -p cm
password> cm
mysql> SELECT * FROM entity;    # or any of the other tables ...
```

2.4 Registering entities

To register an entity, the NSGI-9 REST request 'registerContextRequest' is used.

2.4.1 Request

The REST path of a register request has the form (example using curl):

```
% curl host:port/ngsi9/registerContext --request POST --header
'Content-Type: text/xml' -d <data>
```

The data part of a register request is in XML form and the structure of the data is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
  <contextRegistrationList>
    <contextRegistration>
      <entityIdList>
        <entityId type="" isPattern="false/true">
          <id></id>
        </entityId>
        <entityId type="" isPattern="false/true">
```

```
            <id></id>
          </entityId>
        </entityIdList>
        <contextRegistrationAttributeList>
          <contextRegistrationAttribute>
            <name></name>
            <type></type>
            <isDomain></isDomain>
            <metaData>
              <contextMetadata>
                <name></name>
                <type></type>
                <value></value>
              </contextMetadata>
            </metaData>
          </contextRegistrationAttribute>
        </contextRegistrationAttributeList>
        <registrationMetaData>
          <contextMetadata>
            <name></name>
            <type></type>
            <value></value>
          </contextMetadata>
        </registrationMetaData>
        <providingApplication>URI</providingApplication>
      </contextRegistration>
    </contextRegistrationList>
    <duration></duration>
    <registrationId></registrationId>
  </registerContextRequest>
```

This request is used both to register entities and to update entites already registered. When updating an entity, the 'registrationId' field must be filled in with the Registration Id that was output in the entity's initial registration. If updating more than one entity, all entities must have been registered together initially, otherwise the attempt will fail (as some of the entities have an incorrect registration id). In this case, no update will be done, not even to the entites that match the registration id.

If the request is sent with empty registrationId, the iotConfigMgr will see this an an attempt to add new entities and if any of the entities in the request exist, the entire request will fail - nothing will be done.
When updating entities, attributes can be added to the entity, or metadata can be added to an already existing attribute of an entity. Also, registration metadata can be added (the registration already existed, of course - otherwise the update request would fail).

2.4.2 Response

The response of a registerContextRequest is an XML document of the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<registerContextResponse>
  <duration></duration>
  <registrationId>REGISTRATION_ID</registrationId>
</registerContextResponse>
```

Very important here to save the REGISTRATION_ID together with the registered entites as this registration id must be used in consequent updates of the entities registered. The duration is not mandatory in the response, but if it is present, it simply reflects the duration from the request corresponding to the response.

2.5 Discovering Entities

To discover entities, the NSGI-9 REST request 'discoverContextRequest' is used.

2.5.1 Request

The REST path of a discovery request has the form (example using curl):

```
 % curl host:port/ngsi9/discoverContextAvailability --request POST --header 'Content-Type: text/xml' -d <data>
```

The data part of a register request is in XML form and the structure of the data is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityRequest>
  <entityIdList>
    <entityId type="" isPattern="true/false">
      <id></id>
    </entityId>
    <entityId type="" isPattern="true/false">
      <id></id>
    </entityId>
  </entityIdList>
```

```
  <attributeList>
    <attribute>temperature</attribute>
    <attribute>occupancy</attribute>
    <attribute>lightstatus</attribute>
  </attributeList>
  <restriction>
    <attributeExpression></attributeExpression>
    <scope>
      <operationScope>
        <scopeType></scopeType>
        <scopeValue></scopeValue>
      </operationScope>
      <operationScope>
        <scopeType></scopeType>
        <scopeValue></scopeValue>
      </operationScope>
    </scope>
  </restriction>
</discoverContextAvailabilityRequest>
```

An entity is identified by the combination of its type and id.

2.5.2 Response

The response of a discoverContextAvailabilityRequest is an XML document of the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityResponse>
  <contextRegistrationResponseList>
    <contextRegistrationResponse>
      <contextRegistration>
        <entityIdList>
          <entityId type="Room" isPattern="false">
            <id>OfficeRoom</id>
          </entityId>
          <entityId type="Room" isPattern="false">
            <id>OfficeRoom</id>
          </entityId>
        </entityIdList>
```

```
        <contextRegistrationAttributeList>
          <contextRegistrationAttribute>
            <name>temperature</name>
            <type>degree</type>
            <isDomain>false</isDomain>
            <metaData>
              <contextMetadata>
                <name>ID</name>
                <type>string</type>
                <value>1110</value>
              </contextMetadata>
              <contextMetadata>
                <name></name>
                <type></type>
                <value></value>
              </contextMetadata>
            </metaData>
          </contextRegistrationAttribute>
        </contextRegistrationAttributeList>
        <registrationMetaData>
          <contextMetadata>
            <name>ID</name>
            <type>string</type>
            <value>2212</value>
          </contextMetadata>
          <contextMetadata>
            <name></name>
            <type></type>
            <value></value>
          </contextMetadata>
        </registrationMetaData>
        <providingApplication>http://192.168.100.1:70/application
        </providingApplication>
      </contextRegistration>
    </contextRegistrationResponse>
  </contextRegistrationResponseList>
  <errorCode>
```

```
    200
    <reasonPhrase>Ok</reasonPhrase>
    <details>a</details>
  </errorCode>
</discoverContextAvailabilityResponse>
```

On failure, the entire contextRegistrationResponseList part is omitted.
On success, the errorCode part is omitted (here I have a bug in iotConfigMgr - on success I reply with both parts ...).

2.6 Connecting to iotConfigMgr

This executable is implemented in 'C', and this code example to connect to it is in 'C' as well. It is easy enough to find connect examples in any language.

```
int serverConnect(const char* host, unsigned short port)
{
  int                 fd;
  struct hostent*     hp;
  struct sockaddr_in  peer;
  if ((hp = gethostbyname(host)) == NULL)
  {
    printf("gethostbyname(%s): %s\n", host, strerror(errno));
    return -1;
  }
  if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
  {
    printf("socket: %s\n", strerror(errno));
    return -1;
  }
  memset((char*) &peer, 0, sizeof(peer));
  peer.sin_family      = AF_INET;
  peer.sin_addr.s_addr = ((struct in_addr*) (hp->h_addr))->s_addr;
  peer.sin_port        = htons(port);
  if (connect(fd, (struct sockaddr*) &peer, sizeof(peer)) == -1)
  {
    close(fd);
    printf("connect(%s, %d): %s\n", host, port, strerror(errno));
    return -1;
  }
```

```
   return fd;
 }
```

2.7 Sending requests to iotConfigMgr

In a 'C' program, a normal 'write' is used to send REST commands to the iotConfigMgr. This 'write' will need to follow the REST standard but it is out of the scope of this document to explain REST in detail. From command line, using the tool 'curl', this is how to interact with iotConfigMgr, Examples of this are found above, both for 'register' and 'discover'.
To update a registration, the registration identifier that was output in the initial regiostration must be provided, otherwise the operation will fail.

2.8 Debugging iotConfigMgr

The iotConfigMgr executable maintains a log file at /tmp/iotConfigMgrLog. The more verbose/trace asked at starting iotConfigMgr, the more content the logfile will have. There are a number of command line options to turn on verbosity and trace levels:

- -v
- -vv
- -vvv
- -vvvv
- -vvvvv
- -t <t1-t2,t3-t4>
- -r
- -w

All these verbose/trace levels can also be altered at runtime, using a REST interface:

```
curl --request PUT    host:port/log/verbose/set/3
curl --request PUT    host:port/log/trace/set/0-5,19.34
curl --request PUT    host:port/log/reads/on
curl --request DELETE host:port/log/reads
```

To debug the internal lists of iotConfigMgr, another REST request is defined:

```
curl host:port/debug/entity
curl host:port/debug/registration
curl host:port/debug/attribute
curl host:port/debug/metadata
curl host:port/debug/regMetadata
```

The output of these commands go to the logfile (/tmp/iotConfigMgrLog).
A useful command to view the logfile:

```
% tail -f /tmp/iotConfigMgrLog
```

3 Backend Things Management - IoT Broker User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

3.1 Introduction

Welcome the IoT Broker User and Programmer Guide. The IoT Broker is the component contributed by NEC to the Things Management Generic Enabler implementation by TID/NEC. It is built on a OSGI solution using standard interface NGSI 9/10 to comunicate with the other components/GEs. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up-to-date information on this interface.

3.2 User Guide

The NGSI-10 reference (OMA_NGSI-10) describes how to use the NGSI-10 API in detail. The following sections will go into more detail on how to use the Iot Broker as a user or developer.

3.3 Developer Guide

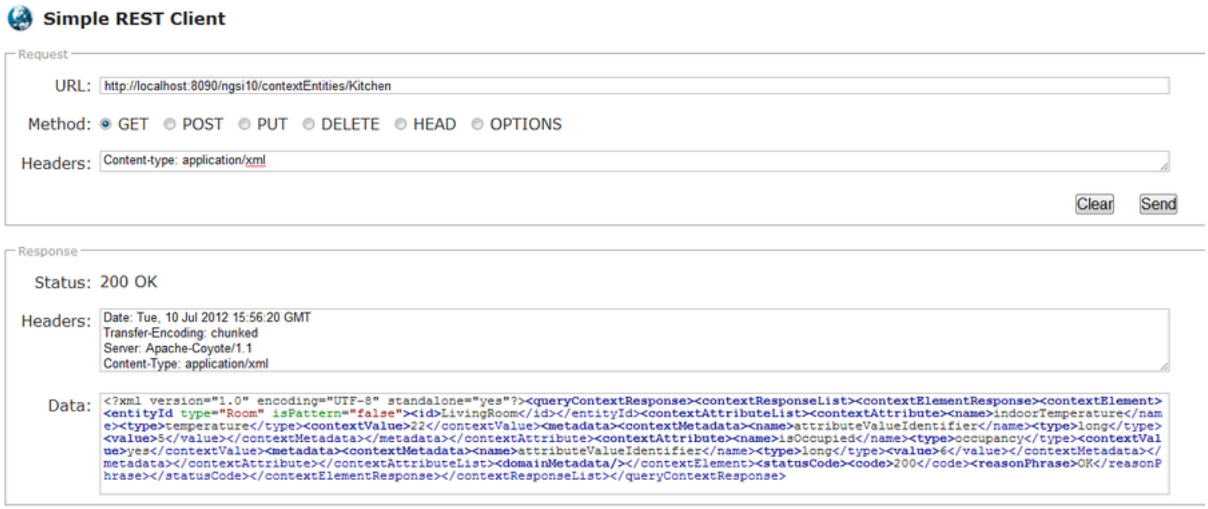
The IoT Broker exposes tha NGSI-10 interface, which is a RESTful API via HTTP. This also means that most programming languages can be used to access the IoT Broker.
To give a feeling of how NGSI-10 works, let us take a look at an HTTP request and the corresponding response:

```
<GET /ngsi10/contextEntities/Kitchen HTTP/1.1
<Host: localhost:80
<User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0)
Gecko/20100101 Firefox/13.0.1
<Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
<Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
<Accept-Encoding: gzip, deflate
<Connection: keep-alive
>
<HTTP/1.1 200 OK
<Server: Apache-Coyote/1.1
<Content-Type: application/xml
<Transfer-Encoding: chunked
<Date: Tue, 10 Jul 2012 15:46:49 GMT
```

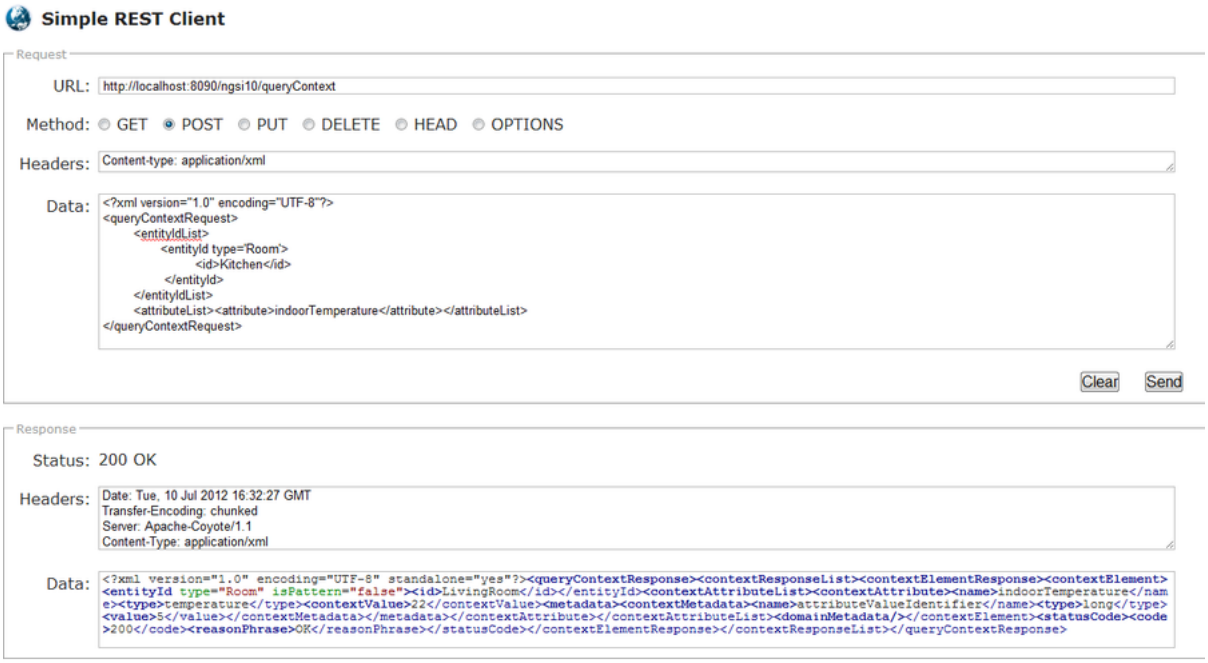
In this example the HTTP GET operation is used.
The NGSI-10 API deals with only one media type, application/xml, which means that we can send only xml content and receive only xml responses.

3.3.1 Accessing the IoT Broker NGSI-10 Interface from a Browser

The following example interactions can be executed using the Chrome browser [1] with the Simple REST Client plugin [2] in order to send http commands to the IoT broker. You can use it also in Firefox through RESTClient add-ons [3].
We give two different example for the GET and POST request:
1. GET request:



2. POST request:


Simple REST Client
Request
URL: http://localhost:8090/ngsi10/queryContext
Method: GET POST PUT DELETE HEAD OPTIONS
Headers: Content-type: application/xml
Data:
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
<entityIdList>
<entityId type='Room'>
<id>Kitchen</id>
</entityId>
</entityIdList>
<attributeList><attribute>indoorTemperature</attribute></attributeList>
</queryContextRequest>
Clear
Send
Response
Status: 200 OK
Headers:
Date: Tue, 10 Jul 2012 16:32:27 GMT
Transfer-Encoding: chunked
Server: Apache-Coyote/1.1
Content-Type: application/xml
Data:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><queryContextResponse><contextResponseList><contextElementResponse><contextElement><entityId type="Room" isPattern="false"><id>LivingRoom</id></entityId><contextAttributeList><contextAttribute><name>indoorTemperature</name><type>temperature</type><contextValue>22</contextValue><metadata><contextMetadata><name>attributeValueIdentifier</name><type>long</type><value>5</value></contextMetadata></metadata></contextAttribute></contextAttributeList><domainMetadata/></contextElement><statusCode><code>200</code><reasonPhrase>OK</reasonPhrase></statusCode></contextElementResponse></contextResponseList></queryContextResponse>

4 Gateway Device Management - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

4.1 Introduction

Welcome the User and Programmer Guide for the Gateway Device Management Generic Enabler. The Gateway Device Management Generic Enabler is responsible for the communication with the Backend and IoT and non-IoT devices. The Gateway Device Management GE includes the functional components to handle the registration/connection phases towards the Backend/Platform, to translate the incoming data or messages in an internal format and to send the outgoing data or messages compliant with IETF CoRE specifications. It is also capable of managing the communication with the IoT Resources, i.e. the devices connected to the IoT Gateway (that may be online or offline), and resources hosted by the gateway. The GE also contains Resource Management capabilities, i.e. to keep track of IoT Resource descriptions that reflect those resources that are reachable via the gateway. These can be both IoT Resources, or resources hosted by legacy devices that are exposed as abstracted IoT Resources. In addition, any IoT resource that is hosted on the gateway itself if also managed by this GE. The GE makes it possible to publish resources in the gateway, and also for the backend to discover what resources are actually available from the gateway.
The generic enabler implements IETF CoRE standards for embedded web services. For the first release of Fiware only devices that support the IETF CoRE CoAP protocol will be supported. CoAP is essentially a stripped down asynchronous version of HTTP designed for constrained devices. This guide will explain the API that developers who want to connect to gateway can implement to access IoT resources in CoAP devicesa and query the resource directory on the gateway.

4.2 Developer Guide

4.2.1 HTTP-CoAP mapping

The gateway exposes IoT resources to the Internet as web resources based on the IETF CoRE CoAP specification. CoAP supports a limited subset of HTTP functionality, and thus a mapping to HTTP is required. IETF provides some drafts which defines how the mapping has to be done. Basically there are two drafts, the “Constrained Application Protocol” (draft-ietf-core-coap-08) and the “Best practices for HTTP-CoAP mapping implementation” (draft-castellani-core-http-mapping-02).

4.2.1.1 *URL address translation*

The CoAP URL is embedded in the HTTP URL. According to this mode of operation an HTTP address:

```
http://authority/coap/SA_ADDR:{port}/resource
```

where authority is the ip address of the gateway and SA_ADDR is the ip address of the device is translated to:

```
coap://SA_ADDR:{port}/resource
```

The CoAP port is optional. If the port is not defined in the URL, the HTTP-Proxy will use the port 5683. This is the default port used in the Californium CoAP stack (http://people.inf.ethz.ch/mkovatsc/californium.php). The HTTP port of the HTTP-CoAP proxy is 8080. An example of a request from a HTTP client to the HTTP-CoAP proxy looks like this:

```
GET http://localhost:8080/coap/localhost/helloWorld
```

4.2.1.2 *Asynchronous messages*

The current HTTP-CoAP implementation on the gateway does not currently support asynchronous messages. That means the client should only expect to receive request/response type of operations. Therefore, subscriptions through this interface are currently not supported.

CoAP Code	CoAP Description	HTTP Code
65	2.01 Created	Created 201
66	2.02 Deleted	OK 200
67	2.03 Valid	OK 200
68	2.04 Changed	OK 200
69	2.05 Content	OK 200
128	4.00 Bad Request	Bad Request 400
129	4.01 Unauthorized	Unauthorized 401
130	4.02 Bad Option	Method Not Allowed 405
131	4.03 Forbidden	Forbidden 403
132	4.04 Not Found	Not Found 404
133	4.05 Method Not Allowed	Method Not Allowed 405
134	4.06 Not Acceptable	Not Acceptable 406
140	4.12 Precondition Failed	Method Not Allowed 405
141	4.13 Request Entity Too Long	Request Entity Too Long 414
143	4.15 Unsupported Media Type	Unsupported Media Type 415
160	5.00 Internal Server Error	Internal Server Error 500
161	5.01 Not Implemented	Not Implemented 501
162	5.02 Bad request	Bad gateway 502
163	5.03 Service Unavailable	Service Unavailable 503
164	5.04 Gateway Timeout	Gateway Timeout 504
165	5.05 Proxying Not Supported	Unauthorize 401

4.2.1.3 *Error handling*

If the content-type of the CoAP message is empty, the HTTP-CoAP sends an 'Unsupported Media Type 415' code to the HTTP client. If the response code from the CoAP message is unknown, the HTTP-CoAP sends a 'Bad Request 400' code to the HTTP client. If the method sent by the HTTP client is unknown, the HTTP-CoAP sends a 'Method not allowed 405' The HTTP methods Connect, Trace, Options returns a 'Not implemented 501'

4.2.1.4 *Content-types*

The HTTP-CoAP implements the CoAP messages with content type 'text/plain; charset=utf-8'. The messages with content types 'application/link-format', 'application/xml', 'application/octet-stream', 'application/exi', and 'application/json' are sent forward to the client without using any parsing.

4.3 Accessing resources

4.3.1 Request for sensor data

```
HTTP GET http://authority/coap/SA_ADDR/resource

Parameters: None

Body: Empty

Response

The full list of the HTTP response codes will depend on the
corresponding CoAP response codes and the mapping between the CoAP
and HTTP response codes described in the CoAP draft [COAP]. See
section 2.4 for response code mappings.

Example

Request: [HTTP GET]
http://m2m.ericsson.com/coap/2011:DB8::11/gpio/btn

Response: 200 OK, Body: 2
```

4.3.2 Actuation Command

```
Request

[HTTP POST] http://authority/coap/SA_ADDR/resource

Body: Depends on resource

Response

The full list of the response codes will depend on the CoAP response
codes

and the mapping between the CoAP and HTTP response codes described
in
```

the CoAP draft. See above response code mappings.

Example

Request: [HTTP PUT] http://m2m.ericsson.com/coap/2011:DB8::11/lt/ledr/on

Body:1

Response: 200 OK

4.3.3 List resources on a device

HTTP GET http://authority/coap/SA_ADDR/.well-known/core

Parameters: None

Body: Empty

Request: [HTTP GET] http://m2m.ericsson.com/coap/2011:DB8::11/.well-known/core

Header: GET (T=CON, …)

Uri-Path: ".well-known"

Uri-Path: "core"

The response to the GET to /.well-know/core resource contains a set of links to all the resources hosted at the sensor node. These links are compliant with the CoRE Link Format IETF Draft. The information for each node includes:

- Path
- Resource Type (rt=)
- Interface (if=)
- Whether the resource is observable.

4.4 Resource directory

The Resource Directory includes information about the number and types of sensors and actuators present in the network and their resources. See the resource directory draft from IETF for more details. The Resource Directory API contains the following functions:

4.4.1 RD Discovery

For the examples below only coap URLs are provided. Please see the [HTTP-CoAP mapping] section above for information how to map this to an HTTP URL.
To discover the RD itself, do a query towards coap://<your-host>:<your-port>/.well-known/core?rt=core-rd. Define as content-type "application/link-format" (or no content-type header). A success response is 2.05 Content
Content-type: application/link-format
Payload: </rd>;rt="core-rd"

4.4.2 Registering resources to RD

To register resources to RD, do a query towards coap://<host>:<port>/rd using CoAP POST method (The {rd-base} in the draft is implemented as rd in this GW. POSTing towards ./well-known/core is not supported.
Method: POST
Content-Type: application/link-format
Query-parameters that can be present in a request are lt, h, ins, rt, domain and context. For more details how to use these, see the draft. The RD will store the data for maximum 24 hours. The same interface can be used also for updating the registration. In the payload, normal link-format the following attributes are supported: -rt
-if
-sz
-ct
-title
-anchor
-title*
-rel
In addition, "key" attribute used for secure CoAP messaging is supported.
Response
If success, 2.01 Created
If internal error, 5.00 Internal server error
Option headers (if success):
Location-Path: <unique path to the registration> (e.g. /rd/1234)

4.4.3 Resource Lookup

Discovering resources is supported with domain and ep parameters as defined in the RD draft.
Core Link format parameters can be used to filter the lookups. The following Core Link Format parameters are supported by the gateway for performing lookups:
-rt
-if
-sz
-ct
-title
-anchor
-title*
-rel
Example Lookup #1

```
GET coap://<host>:<port>/rd?ep=myhostfirst (ep defines the endpoint
name based on which the search is done)


COAP [Request ]

Remote socket address [/127.0.0.1:33452]

Request URI [coap://127.0.0.1:33452/rd]
```

```
Message ID [62473]
Message type [1, Non-Confirmable]
Message code [1]
Option [Content-Type] value [application/link-format]
Option [Uri-Path] value [rd]
Option [Token] value [00]
Option [Uri-Query] value [ep=myhostfirst]

COAP [Response]
Remote socket address [/127.0.0.1:33452]
Message ID [62473]
Message type [1, Non-Confirmable]
Message code [69]
Option [Content-Type] value [text/plain; charset=utf-8]
Option [Token] value [00]
Payload
[<coap://myhost/sensors/light>;if="sensor";rt="LightLux";ct=41,<coap
://myhost/sensors/temp>;if="sensor";rt="TemperatureC";ct=41]
```

Example lookup #2

```
GET coap://<host>:<port>/rd?d=default (d defines the domain name
based on which the search is done)


COAP [Request ]
Remote socket address [/127.0.0.1:33452]
Request URI [coap://127.0.0.1:33452/rd]
Message ID [62474]
Message type [1, Non-Confirmable]
Message code [1]
Option [Content-Type] value [application/link-format]
Option [Uri-Path] value [rd]
Option [Token] value [01]
Option [Uri-Query] value [d=default]

COAP [Response]
Remote socket address [/127.0.0.1:33452]
Message ID [62474]
Message type [1, Non-Confirmable]
```

```
Message code [69]
Option [Content-Type] value [text/plain; charset=utf-8]
Option [Token] value [01]
Payload
[<coap://myhost/sensors/light>;if="sensor";rt="LightLux";ct=41,<coap
://myhost/sensors/temp>;if="sensor";rt="TemperatureC";ct=41]
```

Example lookup #3

```
GET /rd (request without query parameters)


COAP [Request ]
Remote socket address [/127.0.0.1:33452]
Request URI [coap://127.0.0.1:33452/rd]
Message ID [62475]
Message type [1, Non-Confirmable]
Message code [1]
Option [Content-Type] value [application/link-format]
Option [Uri-Path] value [rd]
Option [Token] value [02]

COAP [Response]
Remote socket address [/127.0.0.1:33452]
Message ID [62475]
Message type [1, Non-Confirmable]
Message code [69]
Option [Content-Type] value [text/plain; charset=utf-8]
Option [Token] value [02]
Payload
[<coap://myhost/sensors/light>;if="sensor";rt="LightLux";ct=41,<coap
://myhost/sensors/temp>;if="sensor";rt="TemperatureC";ct=41]
```

4.4.4 Registration update

Existing resources can be updated with PUT. To point to the correct registration, set the Uri-Path headers to point to the Location-Path headers from the registration response from the gateway. If success, "2.04 Changed" will be sent back. Parameters that are present in the request will be updated. If payload is present, old payload is overwritten.
Error codes:
If there's Content-type header in the request but not "application/link-format" (40),"4.00 Bad request" will be replied. Also, if there's payload, but the content-type is defined to something else than 40, "4.00 Bad request" will be sent back.

If no registration with the given Uri-Path headers can be found, a "4.04 Not Found" will be replied. If there's an internal error, "5.03 Service Unavailable" response will be sent back.
Example PUT:

```
COAP [Request ]
Remote socket address [/127.0.1.1:5685]
Request URI [coap://127.0.1.1:5685/rd/1060394878]
Message ID [52268]
Message type [1, Non-Confirmable]
Message code [3]
Option [Content-Type] value [application/link-format]
Option [Uri-Path] value [rd]
Option [Uri-Path] value [1060394878]
Option [Token] value [01]
Payload
[</sensors/temp/1>;ct=41;ins="Indoor";rt="TemperatureC";if="sensor",
</sensors/temp/2>;ct=41;ins="Outdoor";rt="TemperatureC";if="sensor",
</sensors/light>;ct=41;rt="LightLux";if="sensor"]


COAP [Response]
Remote socket address [/127.0.1.1:5685]
Message ID [52268]
Message type [1, Non-Confirmable]
Message code [68]
Response description [2.04 Changed]
Option [Token] value [01]
```

4.4.5 Registration removal

RD registrations can be deleted with CoAP DELETE. To point to the correct registration, set the Uri-Path headers to point to the Location-Path headers from the registration response from the gateway. If a registration can be found and deleted with the given data, "2.02 Deleted" will be sent back.
Error codes:
If no registration with the given Uri-Path headers can be found, a "4.04 Not Found" will be replied. In case of some internal error, "5.03 Service Unavailable" response will be sent back.
Example DELETE:

```
(COAP [Request ]
Remote socket address [/127.0.1.1:5685]
Request URI [coap://127.0.1.1:5685/rd/1060394878]
Message ID [52271]
```

```
Message type [0, Confirmable]
Message code [4]
Option [Content-Type] value [application/link-format]
Option [Uri-Path] value [rd]
Option [Uri-Path] value [1060394878]
Option [Token] value [04]
Payload
[]

COAP [Response]
Remote socket address [/127.0.1.1:5685]
Message ID [52271]
Message type [2, Acknowledgement]
Message code [66]
Response description [2.02 Deleted]
Option [Token] value [04]
```

5 Gateway Protocol Adapter (ZPA) - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

5.1 Introduction

The ZPA is an implementation of the Protocol Adapter GE and is the glue between the generic device access API and a ZigBee WSN (Wireless Sensor Network) that is composed by the target device(s). Through the ZPA a client is able to discover devices, get parameter readings and execute commands to actuate them.
The expected behavior of the ZPA is based on these phases:
Device discovery

- When a new device is discovered and gets available, it shall create an instance of GenericDevice class, and register the instance as an OSGi service with the name com.ericsson.deviceaccess.api.GenericDevice.
- When a previously discovered device gets unavailable, it shall unregister the corresponding OSGi service.

Device parameter read
It is possible to get the value of a parameter of a device
Device actuation
For each action in a service that a device supports, it should implement a protocol specific logic and put it as an action in GenericDevice class instance that is registered as OSGi service.
In this document is described how to develop a client of the ZPA.
If the ZPA is installed together with the Gateway Device Management GE the client is actually an Ericsson Gateway client and should be realized following the instructions in the Gateway Device Management - User and Programmers Guide
If the ZPA is installed standalone the client should be realized with the instructions contained in the “Developer Guide” section.

5.2 Developer Guide

The client of the ZPA implements the logic that should be executed when a new device is discovered in the local network, when a parameter value is needed and when an actuation is requested.
In order to develop the client of the ZPA use Eclipse to configure a project based on Declarative Services (DS) specification (that is a component model that simplifies the creation of components that publish and/or reference OSGi Services) to reference the OSGi service.
It needs to provide an XML file containing the DS declarations, these are the steps to follow:

- Create a folder OSGI-INF
- Right click and select “New” and then select “Component Definition”
- Introduce the name of the file xml -> for example client.xml
- Select the class that contains the application logic: EricssonClient class
- Click on the “Finish”
- Copy the following into OSGI-INF/client.xml in your project:

```
	<?xml version="1.0"?>
	<scr:component
xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" configuration-
policy="optional" modified="modified" name="Tester">
	<implementation
class="it.telecomitalia.client.zigbee.pa.ClientZigBeePA" />
	<property name="network.type" value="ZigBee"/>
	<reference name="GenericDevice"
		bind="setGenericDevice"
		unbind="unsetGenericDevice"
		cardinality="1..n"
		interface="com.ericsson.deviceaccess.api.GenericDevice"
		policy="dynamic"/>
	</scr:component>
```

Also we need to add the following line to the bundle manifest:
Service-Component: OSGI-INF/*.xml
This declaration has the implementation and reference nodes. The implementation node provides the name of the class which implements the component. The reference node declares to DS that our component has a dependency on a service. The name attribute is simply an arbitrary string which names the dependency. The bind attribute is the name of a method in the implementation class that will be called by DS when a service becomes available, or in other words, when a GenericDevice service is registered with the Service Registry, DS will obtain a reference to the new service object and supply it to our component using the specified method. Likewise the unbind attribute is the name of a method that will be called by DS when a service we were using becomes unavailable. The interface attribute specifies the name of the interface we depend on. The cardinality controls whether the dependency is optional or mandatory, and whether it is singular or multiple. The possible values are:

- 0..1: optional and singular, "zero or one"
- 1..1: mandatory and singular, "exactly one"
- 0..n: optional and multiple, "zero to many"
- 1..n: mandatory and multiple, "one to many" or "at least one"

Choose mandatory and multiple, which mean that the client can start when a or multiple service/s is/are available. The policy attribute has a value that can be either "static" or "dynamic", and it states whether the component implementation is able to cope with having services dynamically switched. Choose the policy “dynamic”.
Then in the manifest there are these import packages (that are contained in generic.device.access-1.37-SNAPSHOT.jar and in org.apache.commons.logging_1.1.1.v201101211721.jar)

```
	com.ericsson.deviceaccess.api,
	org.apache.commons.logging
```

In the following paragraph it is shown the code of the client of the ZPA described in the “End-to-end testing” section of the Installation and Administration Guide https://forge.fi-

ware.eu/plugins/mediawiki/wiki/fi-ware-private/index.php/Gateway_Protocol_Adapter_-_Installation_and_Administration_Guide#Standalone_installation. This code is commented in the following section.

```
      package it.telecomitalia.client.zigbee.pa;
      /** Copyright (C) 2012 Telecom Italia S.p.A. **/
      import org.apache.commons.logging.Log;
      import org.apache.commons.logging.LogFactory;
      import com.ericsson.deviceaccess.api.GenericDevice;
      import com.ericsson.deviceaccess.api.GenericDeviceAction;
      import
com.ericsson.deviceaccess.api.GenericDeviceActionResult;
      import com.ericsson.deviceaccess.api.GenericDeviceProperties;
      import com.ericsson.deviceaccess.api.GenericDeviceService;
      public class ClientZigBeePA {
       private final static Log log =
LogFactory.getLog(ClientZigBeePA.class);
       protected void setGenericDevice(GenericDevice dev) {
              int currentTarget;
              String URN;
              URN = dev.getURN();
              System.out.println("setGenericDevice dev.getURN()
"+URN);
              if (dev.getService("SwitchPower") != null) {
                     GenericDeviceService service =
dev.getService("SwitchPower");
                     GenericDeviceProperties properties =
service.getProperties();
                     currentTarget =
properties.getIntValue("CurrentTarget");
                     System.out.println("setGenericDevice
currentTarget "+currentTarget);
                     GenericDeviceAction action =
service.getAction("SetTarget");
                      GenericDeviceProperties args =
action.createArguments();
                        if(currentTarget == 0 )
                           args.setIntValue("newTarget", 1);
                        else
                           args.setIntValue("newTarget", 0);
                        try {
```

```
                                GenericDeviceActionResult result =
action.execute(args);
                            if (result == null)
                                System.out.println("result is
null");
                            else {
                                int code = result.getCode();
                                if (result.getCode() == 0) { //
Action executed successfully
                                        // Get key-value pair
object returned as the result
                                        GenericDeviceProperties
actionResult = result.getValue();
                                } else { // Action failed for some
reason.
                                        System.err.println("Failed
in action: " + result.getReason());
                                }
                            }
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
            } else {
                    System.err.println("Device does not support
SwitchPower service");
            }
            if (dev.getService("PowerSensor") != null) {
                    GenericDeviceService service =
dev.getService("PowerSensor");
                    System.out.println("setGenericDevice PowerSensor
");
                    GenericDeviceProperties properties =
service.getProperties();
                    float value =
properties.getFloatValue("CurrentPower");
                    System.out.println("setGenericDevice
CurrentPower "+value);
            } else {
                    System.err.println("Device does not support
PowerSensor service");
            }
```

```
            }
```

On a device discovery event, a GenericDevice object is created and registered as an OSGi service so the client of the Protocol Adapter can be notified. The OSGi service registered, an instance of com.ericsson.deviceaccess.api.GenericDevice, is provided through the setGenericDevice method. The method setGenericDevice is called when a new service is added into the OSGi framework, i.e. in our case, when a new device gets online and is discovered.
Every device discovered may be identified by its unique identifier (the IEEEAdress):

```
      URN = dev.getURN();
```

and it is possible to get the services associated to the device with this instruction:

```
      dev.getService(<name of service>);
```

where <name of service> could be “SwitchPower” or “PowerSensor” A service of the GenericDevice has a certain number of properties and actions that are defined in the Ericsson file, services.xml. The list of properties can be obtained with this instruction:

```
      service.getProperties();
```

An action can be obtained with this instruction:

```
      service.getAction(<name of action>);
```

where <name of action> could be “setTarget” In the services.xml the types of the properties, the list of the actions and their parameters are specified. According to the type of service supported by the GenericDevice, you can perform a read operation of a data and/or an actuation command of a device. In our case the services supported are two: SwitchPower and PowerSensor [see below the sections extracted from the services.xml]

```
      <service name="SwitchPower">
       <description>This service-type enables basic power switching
for embedding devices.
       </description>
        <category>homeautomation.power</category>
        <actions>
             <action name="SetTarget">
               <description>Requests the Power Switch Service
instance output to be driven
                           to the state indicated by 'newTarget'
Value.
               </description>
               <arguments>
                      <parameter name="newTarget" type="Integer">
                      <description>The desired target. Unit: 0 =
power-off state;
                                                          1 =
power-on state.
```

```
                    </description>
                    <min>0</min>
                    <max>1</max>
                    </parameter>
               </arguments>
             </action>
        </actions>
        <properties>
             <parameter name="CurrentTarget" type="Integer">
               <description>The current target. Unit: 0 = power-off
state;
                                                    1 = power-on
state.
               </description>
               <min>0</min>
               <max>1</max>
             </parameter>
       </properties>
    </service>
```

```
    <service name="PowerSensor">
      <description>This service type enables reading a power
sensor.</description>
      <category>homeautomation.power</category>
      <properties>
           <parameter name="CurrentPower" type="Float">
           <description>The current power measured by the sensor.
Unit: Watt.
           </description>
           </parameter>
      </properties>
    </service>
```

SwitchPower has a property “CurrentTarget” and has an action “SetTarget” while the PowerSensor has a property “CurrentPower”. For the first service it is possible to read the “CurrentTarget” property to retrieve the currently value of the status of the device (it can be “on” or “off”):

```
    currentTarget = properties.getIntValue("CurrentTarget");
```

In order to prepare an action it is needed to create the arguments and to set their parameters:

```
GenericDeviceProperties args = action.createArguments();
args.setIntValue("newTarget", 0);
```

and then execute the action “SetTarget” to switch on or off the device:

```
GenericDeviceActionResult result = action.execute(args);
```

For the second service it is possible to read the “CurrentPower” property to get the value of the power measure of the device:

```
float value = properties.getFloatValue("CurrentPower");
```

6 Gateway Data Handling - SOL CEP User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.
[NOTE to project mgmt: this can be moved to public section]

6.1 Introduction

SOL/CEP (Smart Object Lab Complex Event Processor) is driven using a domain specific language called Dolce, which stands for **D**escripti**o**n **L**anguage for **C**omplex **E**vents.
The first version of the Dolce language specification.

6.1.1 Addressed Topics

- *Introduction* to the language. Explains the concepts and how to create complex events from simple events.
- *Types, operators and expressions*, addressing data types, constants, expressions and dedicated complex event functions.
- *Program structure* explores in more detail the construction of complex events, their interaction with the simple events, as well as functionality such as sliding time and tuple windows.

6.1.2 Configuration

The Dolce complex event detection specification is presented to SOL/CEP by means of a text file that must be configured.

6.1.2.1 *Configuration file*

In order for the changes to take place, the configuration file must be edited.
The Application does not specify a standard location for the configuration file.
At startup, it looks for a configuration file named `solcep.conf.xml` in the same directory as from where the application is started.
The location can be overridden by specifinyg `-c <configFile>` as a command line option.

The following section of the configuration must be inspected:

<solcep>
This is the top level of the configuration file.

- **<serverRoot>** Is an absolute, existing path, without a trailing '/'. It indicates the base path for log files and configuration files.

The following sections of the configuration file must be edited:

<complexEventDetector>
Declares a Complex Event Detector with *id*

- <input>
  - <port> The port where the Complex Event Detector listens for decoded events.
  - <specificationFile>The Dolce complex event specification used in the detection of complex event. This file is relative to the **<serverRoot>**

6.1.2.2 *Applying the changes*

The Application needs to be restarted for the changes to be applied. This is achieved with the `/etc/init.d/solcep_ctrl restart` command.

6.1.3 Restrictions

SOL/CEP is a first proof of concept of how Dolce can be implemented. The current implementation is a subset of the specification, and the goal is to have a full reference implementation in the second release of FI-WARE.
SOL/CEP implements the following features of Dolce.

- Event specification
- Complex Event detection
- The `int` data type
- The `and` and `or` event operators
- Time windows (non-recurring)
- Event filtering, allowing for different channels

7 Gateway Data Handling - Esper4FastData Mobile - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

7.1.1 Introduction to CEP Manager engine

7.1.1.1 *Events overview*

Basically, an event is just « something that happens ».
In philosophy, events are objects in time, or instantiations of properties in objects, whereas in computing, they are action that are usually initiated outside the scope of a program and that is handled by a piece of code inside the program.
Complex Event Processing(CEP) is a usual solution for the following questions :

- How to handle massively growing data volumes ?
- how to make meanings of all events flowing through your system at the speed of your business ?
- How to preserve flexibility ?

7.1.1.2 *Value-added of a CEP system*

It takes subsequent action in real time, delivers high-speed processing of many events, operates across all the layers of an organization. It also filters the most meaningful events, merges data from many events and has to deal with event privacy and subscribing.

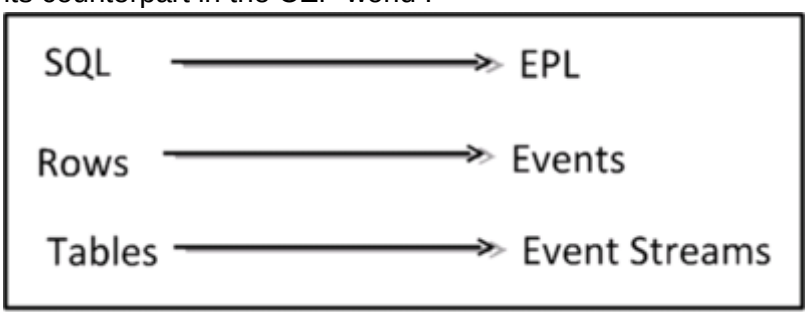
7.1.1.3 *The Esper CEP library*

CEP Mobile Manager engine is based upon the open-source Esper library. There is a very active community of developpers behind it.
Here are some technical facts about it :

- Is available under GPL v2
- Is available as a Java library (jar)
- Has been adapted to Android
- Has tiny footprint

7.1.1.4 *Esper Concepts*

Esper is like a database that has been turned upside-down, every database concept having its counterpart in the CEP world :

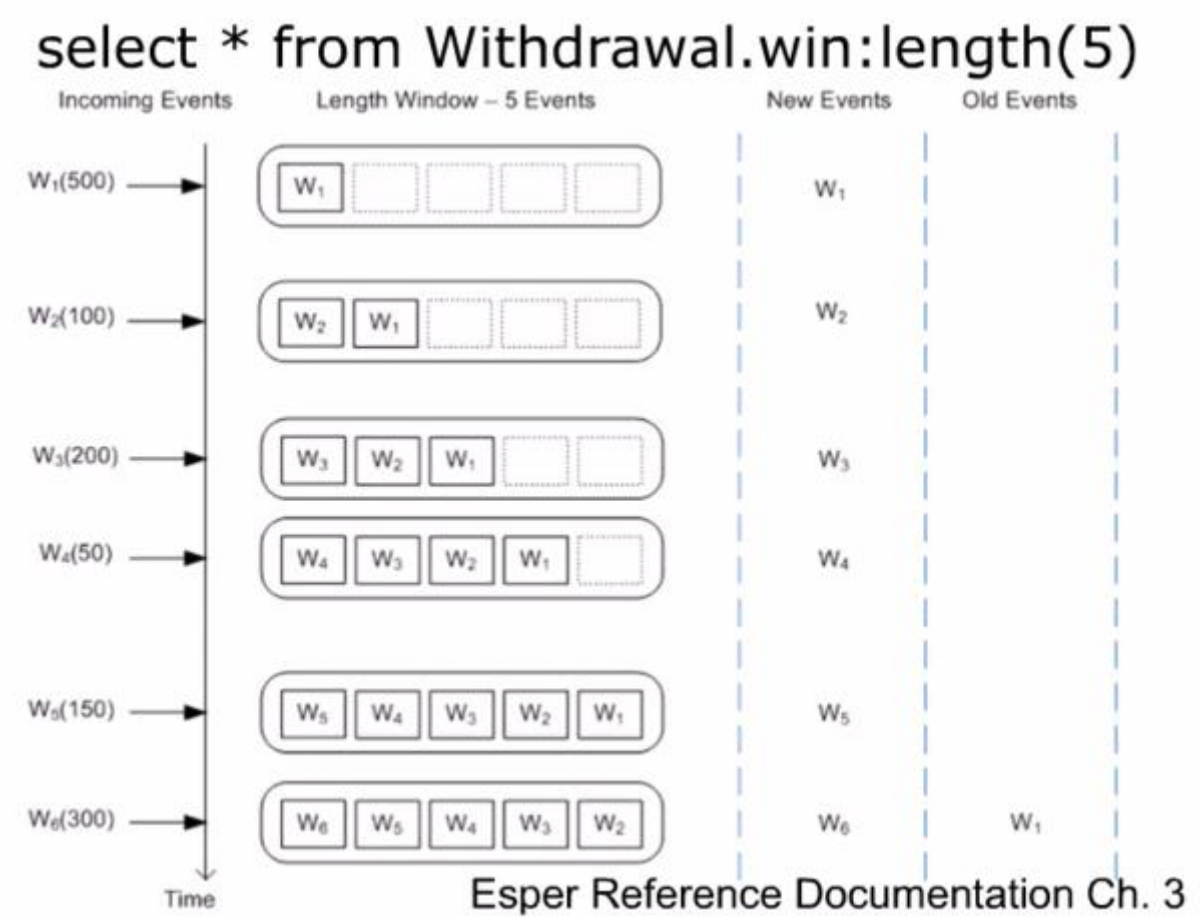


The Esper library is based on the Event Processing Language(EPL), that looks roughly like SQL. But the "querying" model is completely different, because CEP provides a continuous model of querying. This could be named "real-time data mining", in contrary to "store-now query-ater" model, which is equivalent to analyzing historical data.
Despite the syntax similarities between EPL and SQL, CEP is definitely not a database replacement.
Esper uses events windows to store event streams, which can be compared to tables in the database world. These windows have a predefined size, in terms of data volume, or retention time. They are in fact sliding windows that contain the events flowing THROUGH the EPL statement. This is a key point to understand the difference between a database and a CEP :

- In database world, one can instanciate an SQL statement to SYNCHRONOUSLY retrieve particular data
- In the CEP world it's the data that ASYNCHRONOUSLY flows THROUGH an EPL statement, which is trigerred only if properties criteria are fulfiled

7.1.1.5 *Basic EPL statements and sliding events windows*

Filter-free statement
A length window instructs the engine to only keep the last N events for a stream. The next statement applies a length window onto the Withdrawal event stream. The statement serves to illustrate the concept of data window and events entering and leaving a data window:
select * from Withdrawal.win:length(5)
The size of this statement's length window is five events. The engine enters all arriving Withdrawal events into the length window. When the length window is full, the oldest Withdrawal event is pushed out the window. The engine indicates to listeners all events entering the window as new events, and all events leaving the window as old events.
While the term insert stream denotes new events arriving, the term remove stream denotes events leaving a data window, or changing aggregation values. In this example, the remove stream is the stream of Withdrawal events that leave the length window, and such events are posted to listeners as old events.
The next diagram illustrates how the length window contents change as events arrive and shows the events posted to an update listener :

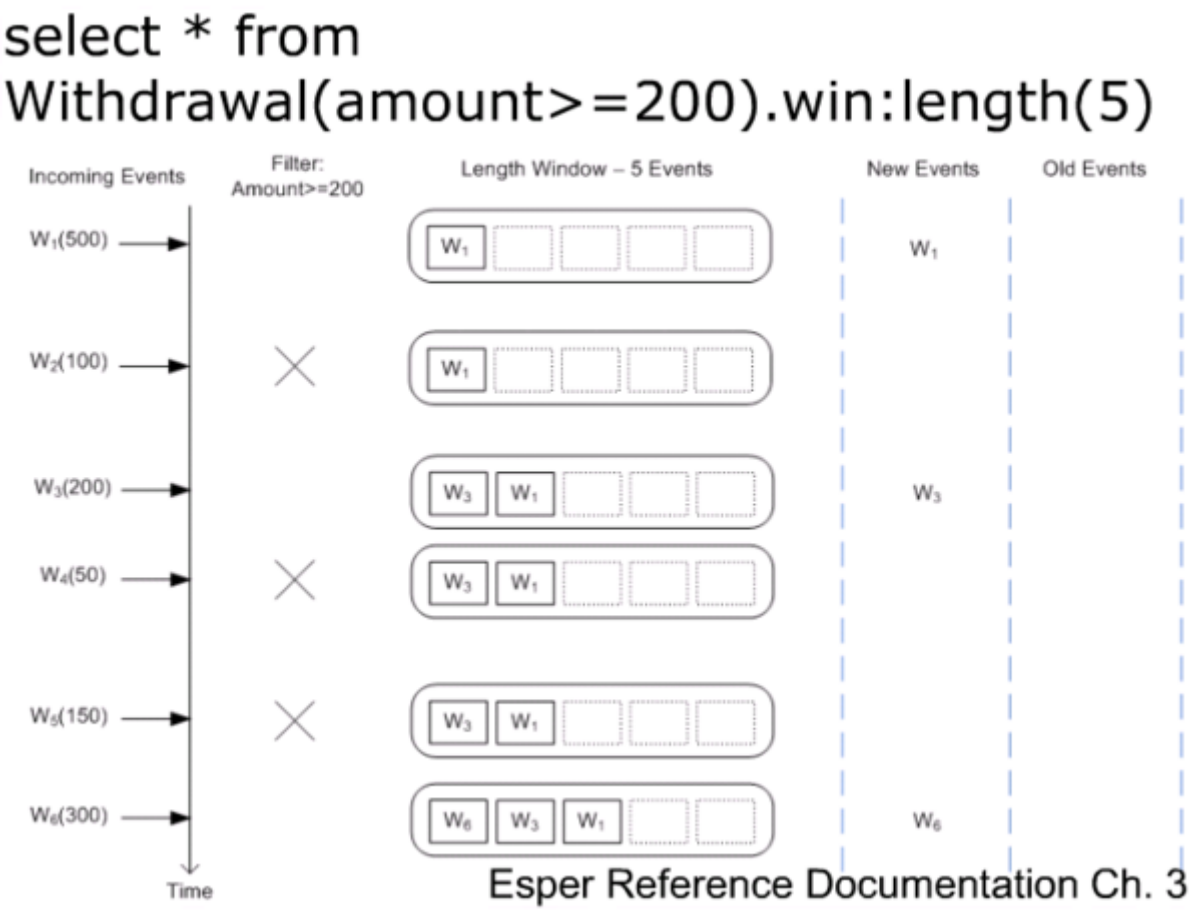


Input filter statement

Filters to event streams allow filtering events out of a given stream before events enter a data window. The statement below shows a filter that selects Withdrawal events with an amount value of 200 or more.

select * from Withdrawal(amount>=200).win:length(5)

With the filter, any Withdrawal events that have an amount of less then 200 do not enter the length window and are therefore not passed to update listeners :



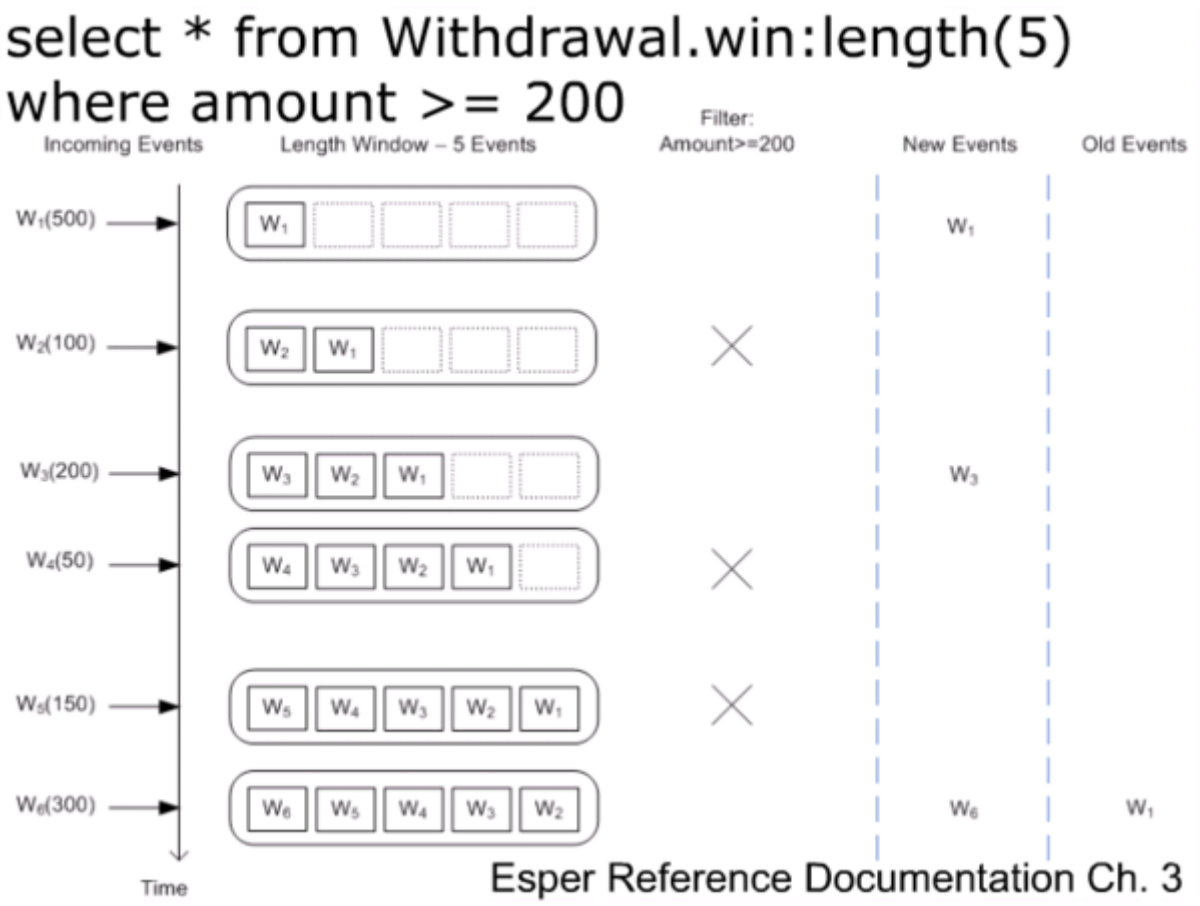
Output filter statement

The where-clause and having-clause in statements eliminate potential result rows at a later stage in processing, after events have been processed into a statement's data window or other views.

The next statement applies a where-clause to Withdrawal events.

```
select * from Withdrawal.win:length(5) where amount >= 200
```

The where-clause applies to both new events and old events. As the diagram below shows, arriving events enter the window however only events that pass the where-clause are handed to update listeners. Also, as events leave the data window, only those events that pass the conditions in the where-clause are posted to listeners as old events.



7.1.1.6 *Detailed EPL Reference*

See http://esper.codehaus.org/esper-4.6.0/doc/reference/en-US/html/epl_clauses.html

7.1.1.7 *Glossary of terms*

- Event: anything that happens, or is contemplated as happening
- Event Object, event message: an object that represents, encore or records an event, generally for the purpose of computer processing
- Event Type: a class of event objects. All events must be instances of an event type. An event has the structure defined by its type. Event types should be defined with a XML Schema Definition (XSD file)
- Event attribute or event property: a component of the structure of an event. An event attribute can have a simple or complex data type.
- Event processing: computing that performs operations on events, including reading, creating, transforming and deleting events.
- Timestamp: a time value attribute of an event. The time in which the event was created:creation time or observed:arrival time.
- Complex event processing (CEP): computing that performs operations on complex events, including reading, creating, transforming or abstracting them.
- Statement or event processing rules: a prescribed method for processing event. Event processing rules are described in Event Processing Language

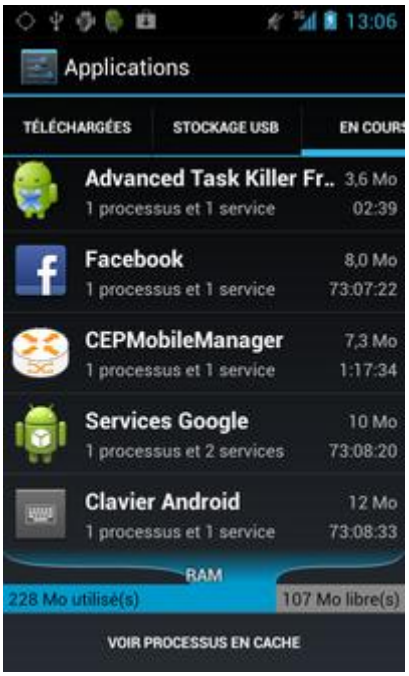
- Event Processing Language: a high level computer language for defining the behavior of event processing agents

7.1.2 Esper4FastData Mobile User and Programmer Guide

This component is a part of the Data Handling Generic Enabler and is a Rest Web service project using standard interface NGSI 9/10 to communicate with the others components/GE. The following sections explains how to use the Esper4FastData Mobile or CEPMobileManager as a user or developer

7.1.2.1 *Accessing the Esper4FastData Mobile*

CEPMobileManager project allows to manage a single shared Complex Event Processing engine on mobile phone. The methods to manage the CEP engine can be accessed and tested thanks to an Android graphic user interface (GUI). In the Android Menu, go to Settings/Applications/Running in order to diplay running processes."CEPMobileManager" should be available in the list. The CEPMobileManager is approachable in the Android Application List.



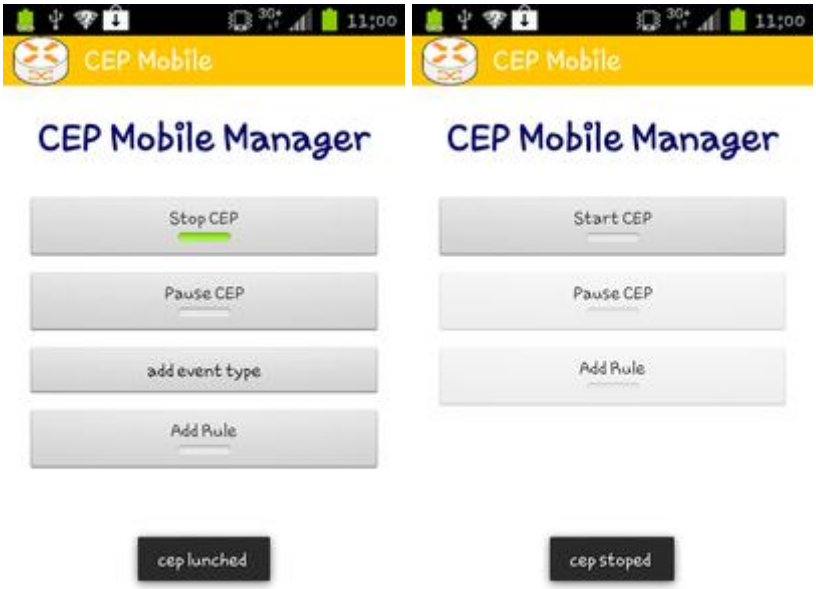
7.1.2.2 *CEP Engine methods*

- start: start the CEP engine if it is not already started

the engine must be started to call all CEPManager methods: adding eventType, rule (statements)…

- stop: stop the running CEP engine

a stopped engine has no more event Types and rules defined

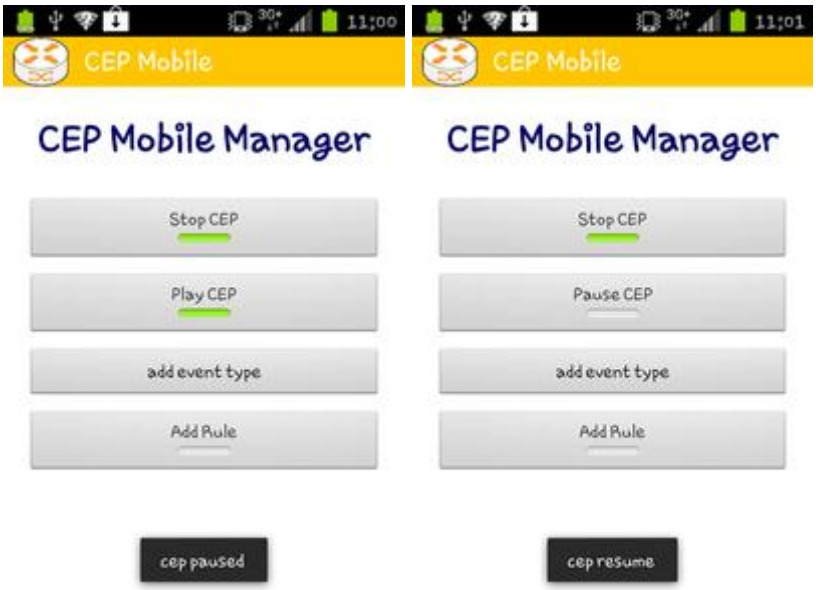


- pause: pause the running CEP engine

Paused engine keeps eventTypes and rules defined

- resume: resume the paused CEP engine

Paused engine keeps eventTypes and rules defined



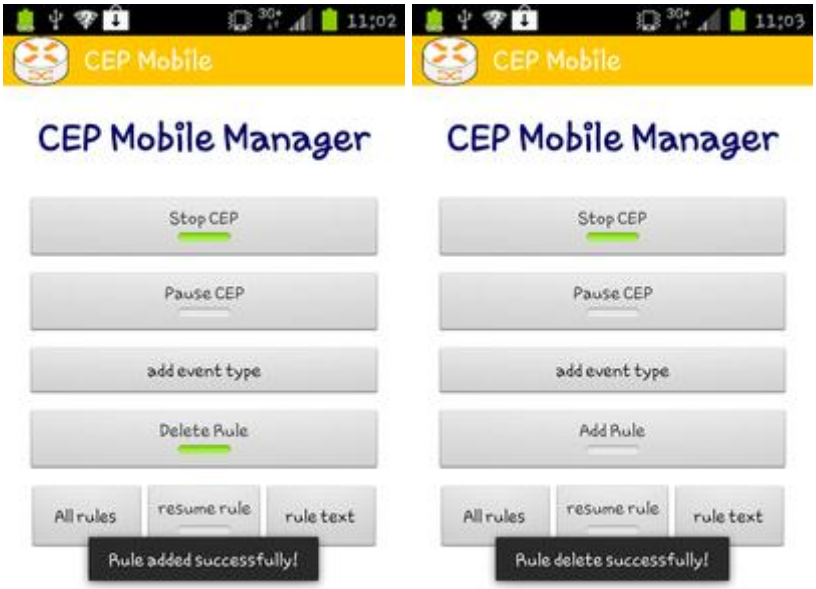
7.1.2.3 *Rule/Statement methods*

- Add Rule: add a rule to the CEP running

Creates and starts an EPL rule. The CEP engine must be started The eventType used in rule must have been defined before adding rule The engine assigns a unique name to the rule. The returned rule is in started state. The rule name is optimally a unique name. If a rule of

the same name has already been created, the engine assigns a postfix to create a unique statement name.

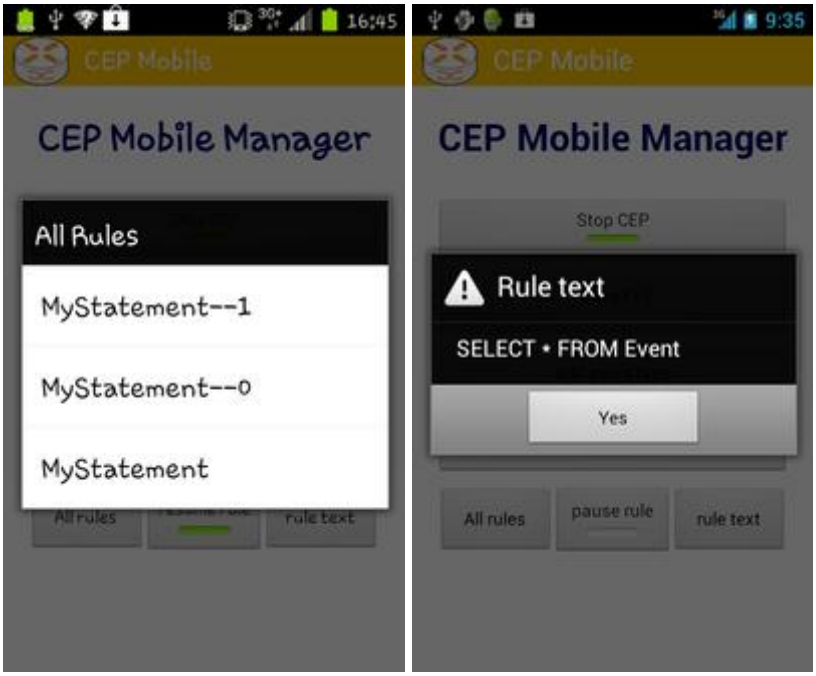
- Delete Rule: delete a specific rule



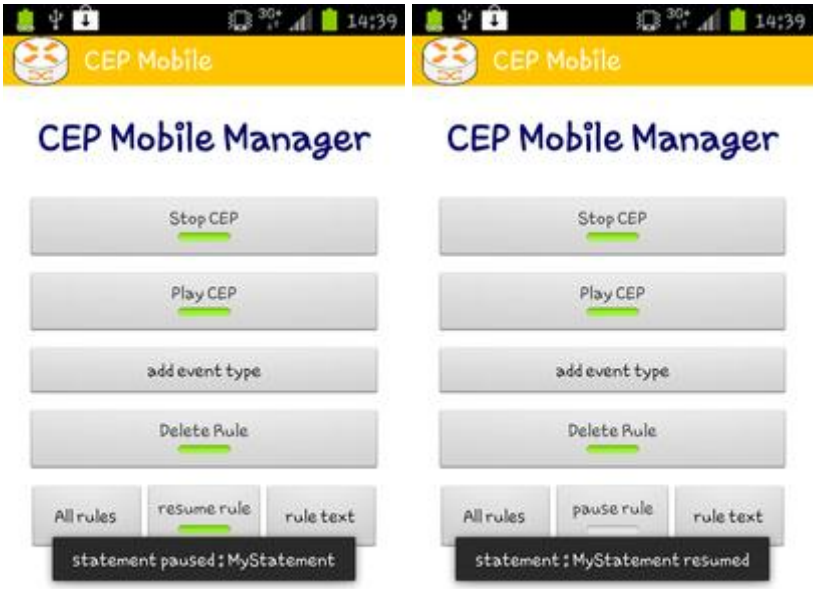
- all Rules: get all names of rules currently deployed in the CEP engine

Returns the rule names of all started and stopped rules. This excludes the name of destroyed rules

- Rule Text: get the rule text for the named rule

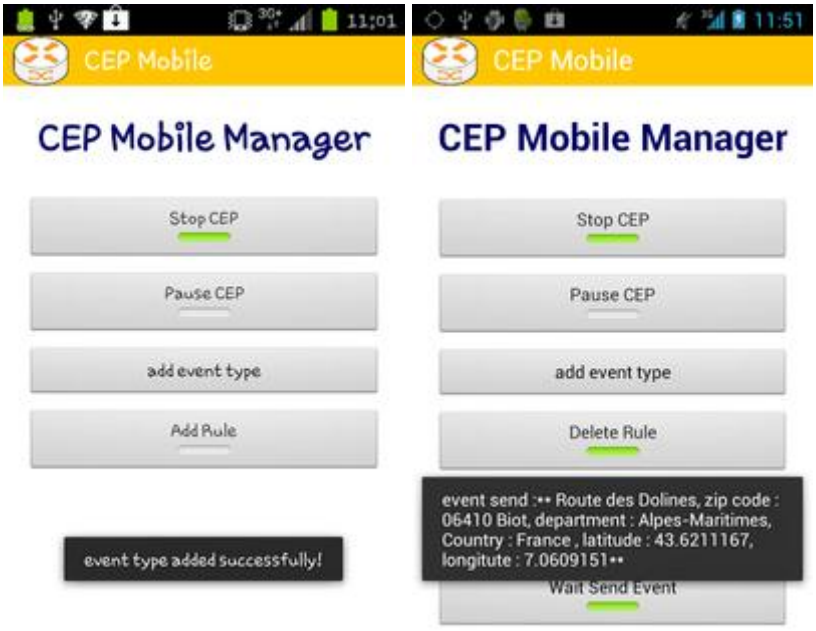


- pause Rule: pause a specific rule
- Resume Rule: resume a specific rule



7.1.2.4 *EventType methods*

- add Event Type: add an event type to the CEP engine
- send Event: send an event within the CEP



8 Gateway Data Handling - Esper4FastData Servlet - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

8.1.1 Introduction to Esper4FastData engine

8.1.1.1 *Events overview*

Basically, an event is just « something that happens ».
In philosophy, events are objects in time, or instantiations of properties in objects, whereas in computing, they are action that are usually initiated outside the scope of a program and that is handled by a piece of code inside the program.
Complex Event Processing(CEP) is a usual solution for the following questions :

- How to handle massively growing data volumes ?
- How to make meanings of all events flowing through your system at the speed of your business ?
- How to preserve flexibility ?

8.1.1.2 *Value-added of a CEP system*

It takes subsequent action in real time, delivers high-speed processing of many events, operates across all the layers of an organization. It also filters the most meaningful events, merges data from many events and has to deal with event privacy and subscribing.

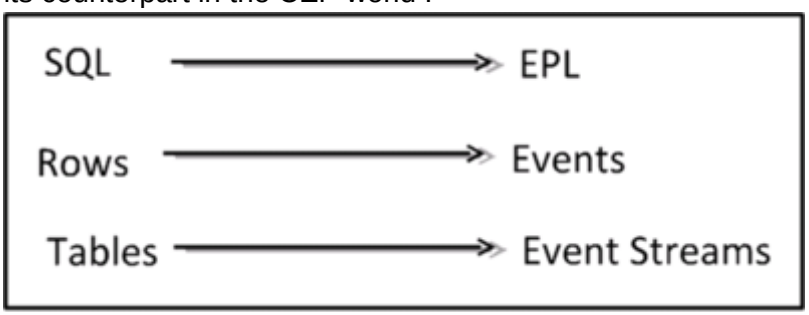
8.1.1.3 *The Esper CEP library*

Esper4FastData engine is based upon the open-source Esper library. There is a very active community of developpers behind it.
Here are some technical facts about it :

- Is available under GPL v2
- Is available as a Java library (jar)
- Has been adapted to Android
- Has tiny footprint

8.1.1.4 *Esper Concepts*

Esper is like a database that has been turned upside-down, every database concept having its counterpart in the CEP world :

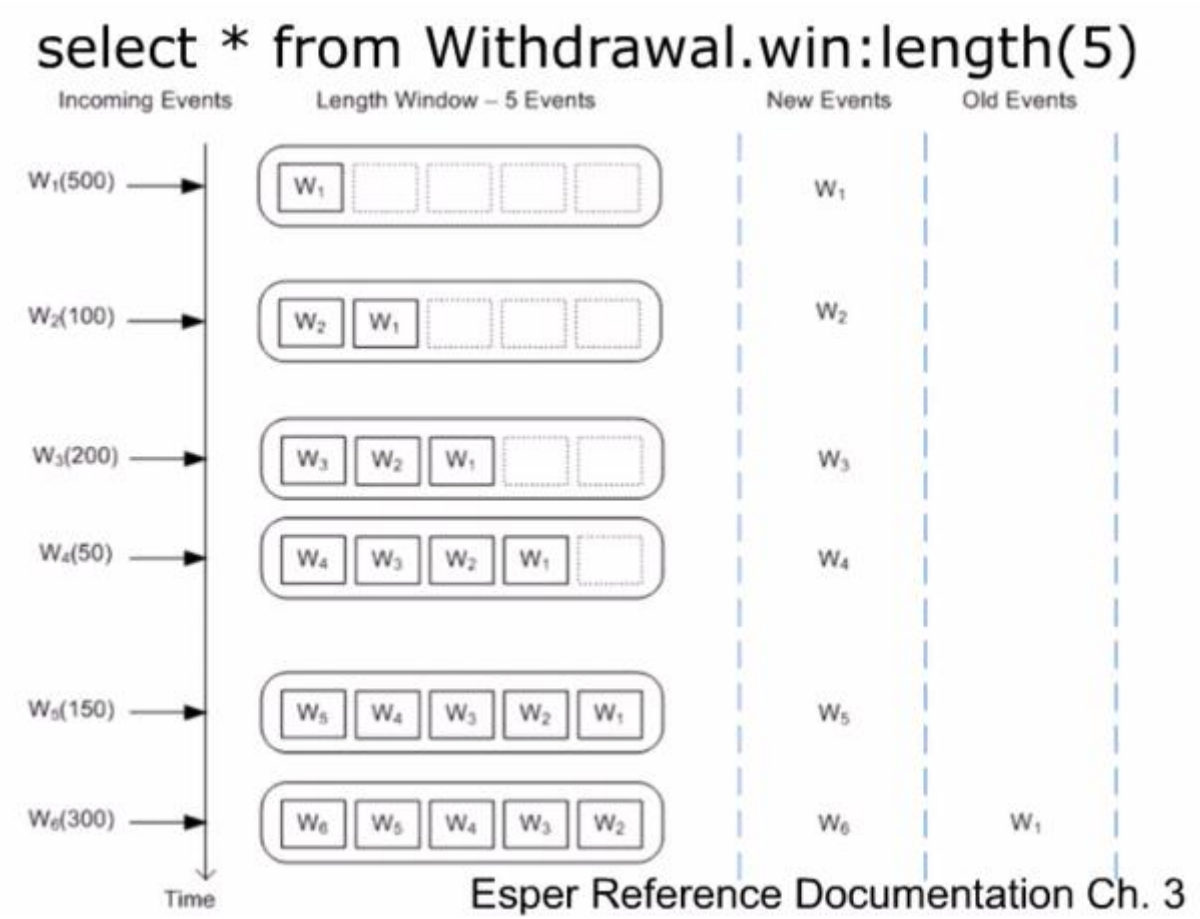


The Esper library is based on the Event Processing Language(EPL), that looks roughly like SQL. But the "querying" model is completely different, because CEP provides a continuous model of querying. This could be named "real-time data mining", in contrary to "store-now query-ater" model, which is equivalent to analyzing historical data.
Despite the syntax similarities between EPL and SQL, CEP is definitely not a database replacement.
Esper uses events windows to store event streams, which can be compared to tables in the database world. These windows have a predefined size, in terms of data volume, or retention time. They are in fact sliding windows that contain the events flowing THROUGH the EPL statement. This is a key point to understand the difference between a database and a CEP :

- In database world, one can instanciate an SQL statement to SYNCHRONOUSLY retrieve particular data
- In the CEP world it's the data that ASYNCHRONOUSLY flows THROUGH an EPL statement, which is trigerred only if properties criteria are fulfiled

8.1.1.5 *Basic EPL statements and sliding events windows*

Filter-free statement
A length window instructs the engine to only keep the last N events for a stream. The next statement applies a length window onto the Withdrawal event stream. The statement serves to illustrate the concept of data window and events entering and leaving a data window:
select * from Withdrawal.win:length(5)
The size of this statement's length window is five events. The engine enters all arriving Withdrawal events into the length window. When the length window is full, the oldest Withdrawal event is pushed out the window. The engine indicates to listeners all events entering the window as new events, and all events leaving the window as old events.
While the term insert stream denotes new events arriving, the term remove stream denotes events leaving a data window, or changing aggregation values. In this example, the remove stream is the stream of Withdrawal events that leave the length window, and such events are posted to listeners as old events.
The next diagram illustrates how the length window contents change as events arrive and shows the events posted to an update listener :

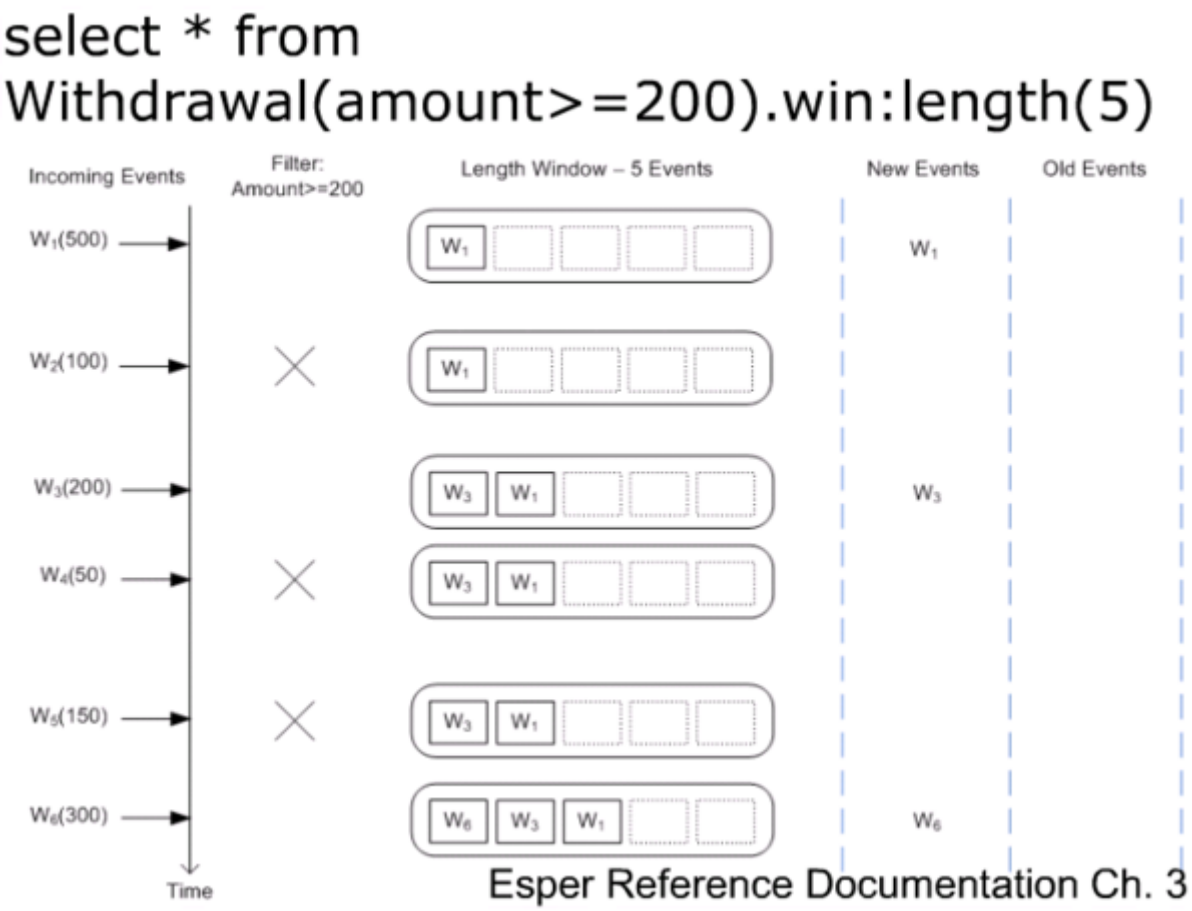


Input filter statement

Filters to event streams allow filtering events out of a given stream before events enter a data window. The statement below shows a filter that selects Withdrawal events with an amount value of 200 or more.

select * from Withdrawal(amount>=200).win:length(5)

With the filter, any Withdrawal events that have an amount of less then 200 do not enter the length window and are therefore not passed to update listeners :



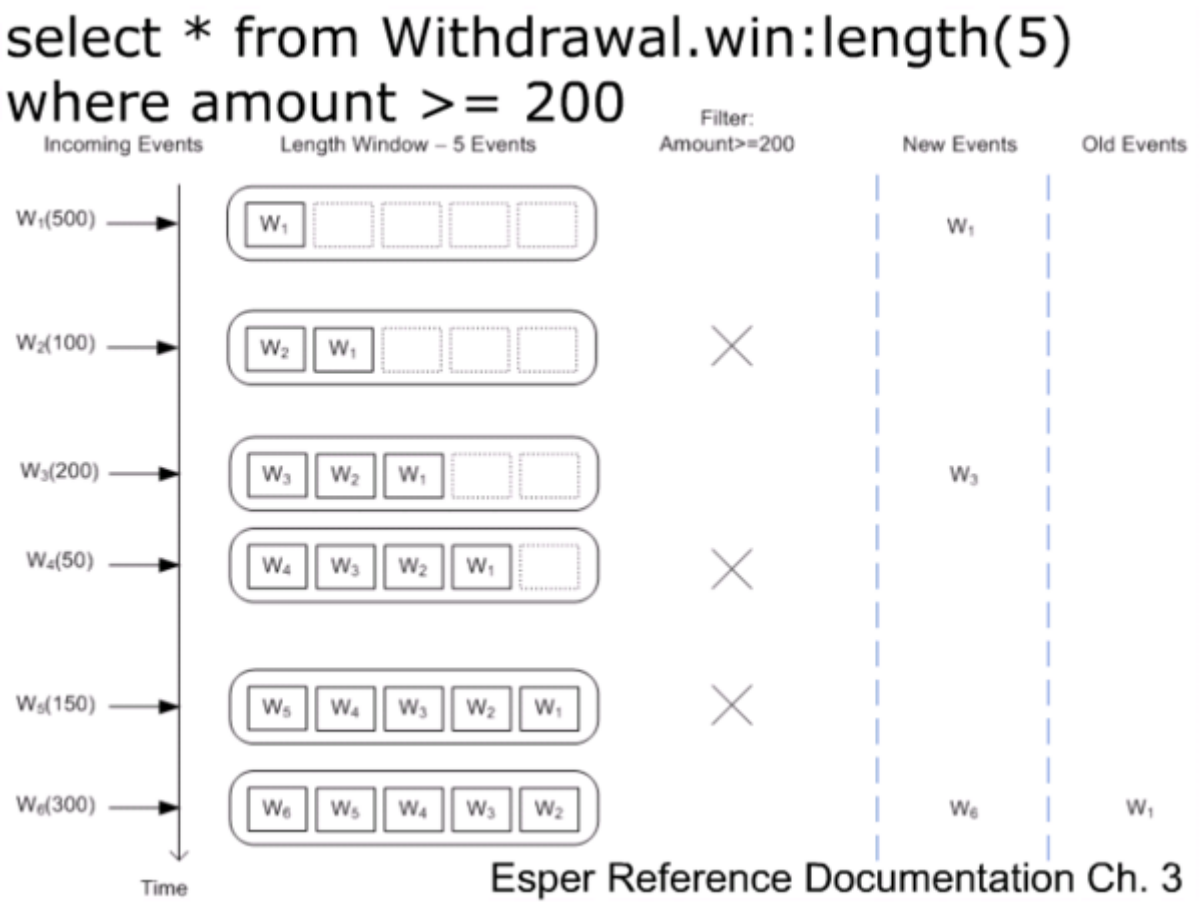
Output filter statement

The where-clause and having-clause in statements eliminate potential result rows at a later stage in processing, after events have been processed into a statement's data window or other views.

The next statement applies a where-clause to Withdrawal events.

```
select * from Withdrawal.win:length(5) where amount >= 200
```

The where-clause applies to both new events and old events. As the diagram below shows, arriving events enter the window however only events that pass the where-clause are handed to update listeners. Also, as events leave the data window, only those events that pass the conditions in the where-clause are posted to listeners as old events.



8.1.1.6 *Detailed EPL Reference*

See http://esper.codehaus.org/esper-4.6.0/doc/reference/en-US/html/epl_clauses.html

8.1.1.7 *Glossary of terms*

- Event: anything that happens, or is contemplated as happening
- Event Object, event message: an object that represents, encore or records an event, generally for the purpose of computer processing
- Event Type: a class of event objects. All events must be instances of an event type. An event has the structure defined by its type. Event types should be defined with a XML Schema Definition (XSD file)
- Event attribute or event property: a component of the structure of an event. An event attribute can have a simple or complex data type.
- Event processing: computing that performs operations on events, including reading, creating, transforming and deleting events.
- Timestamp: a time value attribute of an event. The time in which the event was created:creation time or observed:arrival time.
- Complex event processing (CEP): computing that performs operations on complex events, including reading, creating, transforming or abstracting them.
- Statement or event processing rules: a prescribed method for processing event. Event processing rules are described in Event Processing Language

- Event Processing Language: a high level computer language for defining the behavior of event processing agents

8.2 Standalone CEP Manager User and Programmer Guide

Welcome the CEP Manager User and Programmer Guide. This component is a part of the Data Handling Generic Enabler and is a Rest Web service project using standard interface NGSI 9/10 to comunicate with the others components/GE. The following sections explains how to use the Gateway CEP Manager as a user or developer. The Gateway CEP Manager exposes methods to manage the CEP Engine which is a RESTful API via HTTP.

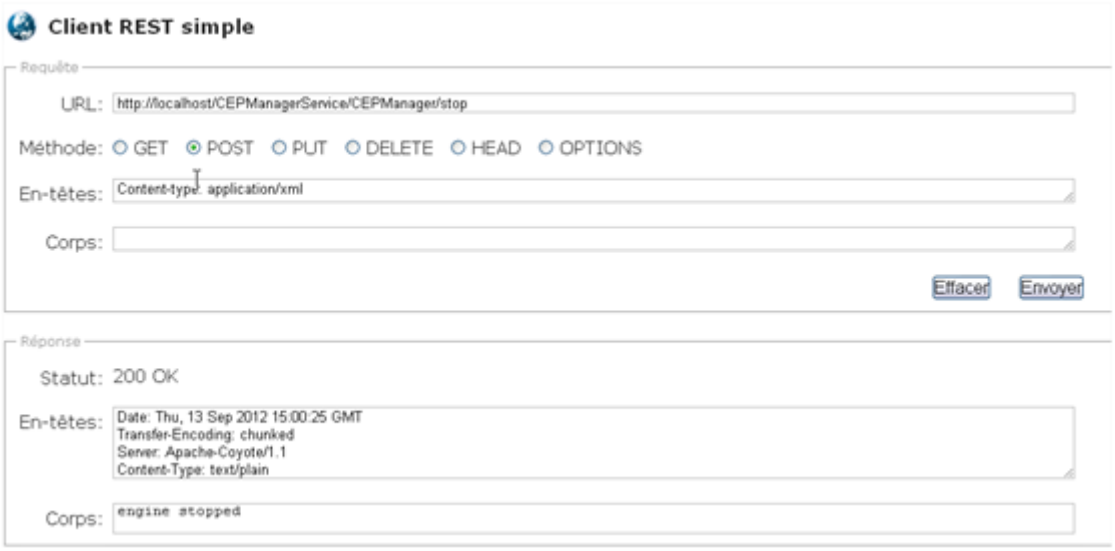
8.2.1 Accessing the Standalone CEP Manager Interface from a Browser

The following example interactions can be executed using the Chrome browser [1] with the Simple REST Client plugin [2] in order to send http commands to the CEP Manager. You can use it also in Firefox through RESTClient add-ons [3].
All examples use the Chrome browser

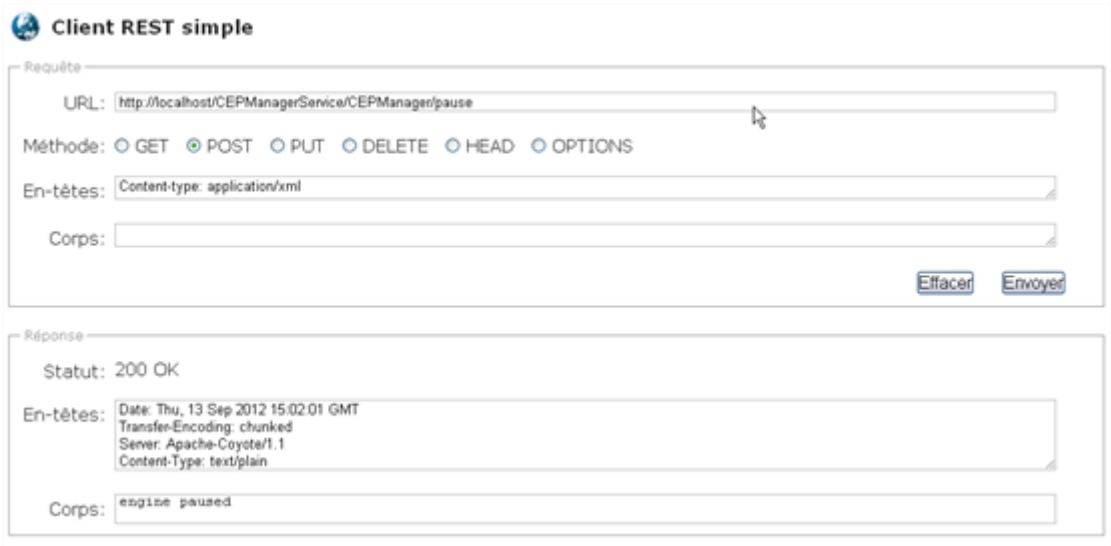
8.2.1.1 *CEP Engine methods*

1. Start POST request: start the CEP engine if it is not already started the engine must be started to call all CEPManager methods: adding eventType, statements…
URL: http://localhost/CEPManagerService/CEPManager/start?engineURI=testEngine

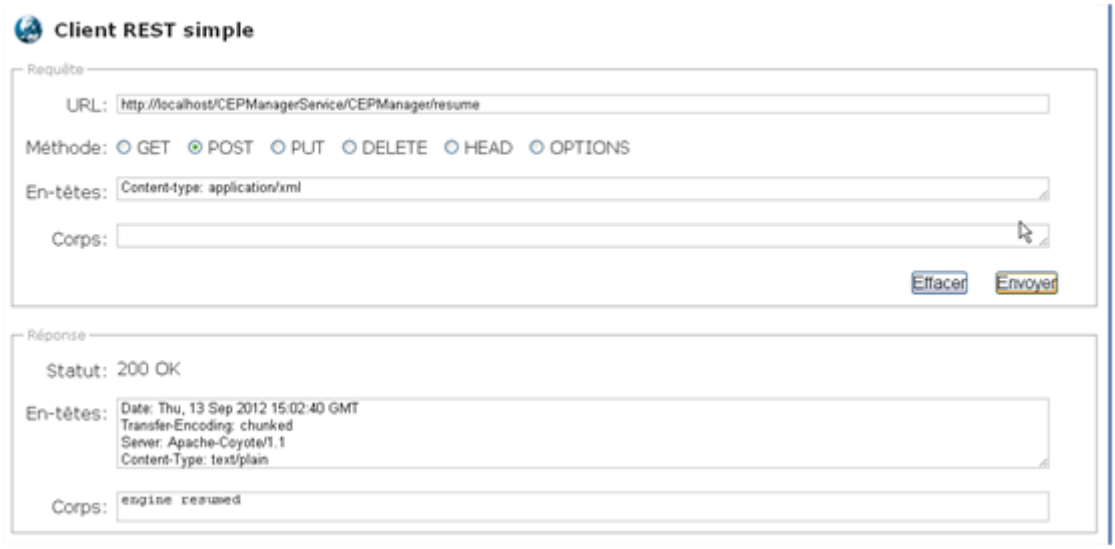
2. Stop POST request: stop the running CEP engine a stopped engine has no more event Types and statements defined
URL: http://localhost/CEPManagerService/CEPManager/stop



3. Pause POST request: pause the running CEP engine Paused engine keeps eventTypes and statements defined
URL: http://localhost/CEPManagerService/CEPManager/pause



4. Resume POST request: resume the paused CEP engine Paused engine keeps eventTypes and statements defined
URL: http://localhost/CEPManagerService/CEPManager/resume



8.2.1.2 *Statement methods*

1. addStatement POST request: add a statement to the CEP running Creates and starts an EPL statement. The CEP engine must be started The eventType used in statement must have been defined before adding statement The engine assigns a unique name to the statement. The returned statement is in started state. The statement name is optimally a unique name. If a statement of the same name has already been created, the engine assigns a postfix to create a unique statement name.
URL:
http://localhost/CEPManagerService/CEPManager/addStatement?statementName=stat0&statement=select+*+from+contextElement

2. getStatementsName GET request: get all names of statements currently deployed in the CEP engine Returns the statement names of all started and stopped statements. This excludes the name of destroyed statements
URL: http://localhost/CEPManagerService/CEPManager/getStatementsName

3. getStatementFields GET request: get attributes selected for a specific statement
URL:
http://localhost/CEPManagerService/CEPManager/getStatementFields?statementName=stat0

4. getStatementState GET request: get the state of a specific statement
URL:
http://localhost/CEPManagerService/CEPManager/getStatementState?statementName=stat0

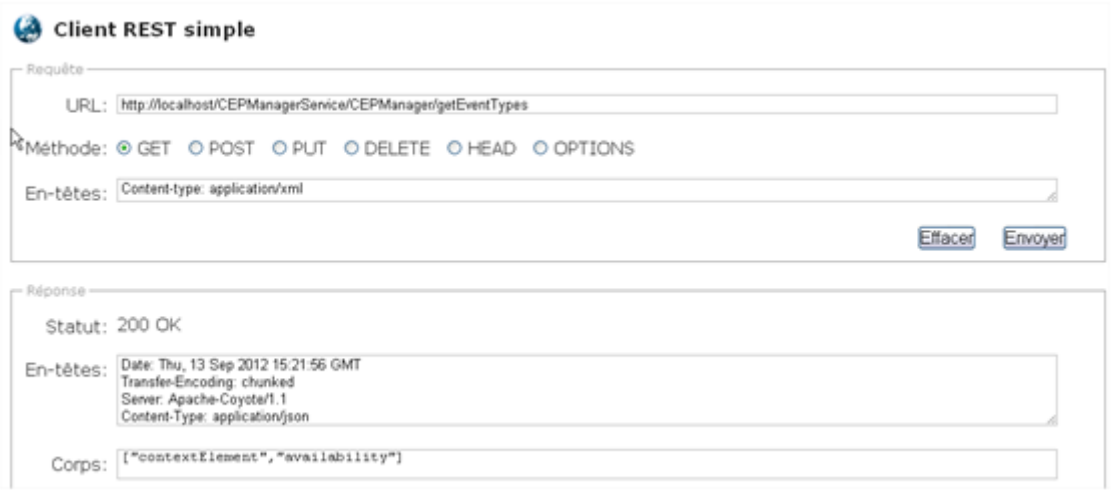
5. pauseStatement POST request: pause a specific statement
URL:
http://localhost/CEPManagerService/CEPManager/pauseStatement?statementName=stat0

6. resumeStatement POST request: resume a specific statement
URL:
http://localhost/CEPManagerService/CEPManager/pauseStatement?statementName=stat0

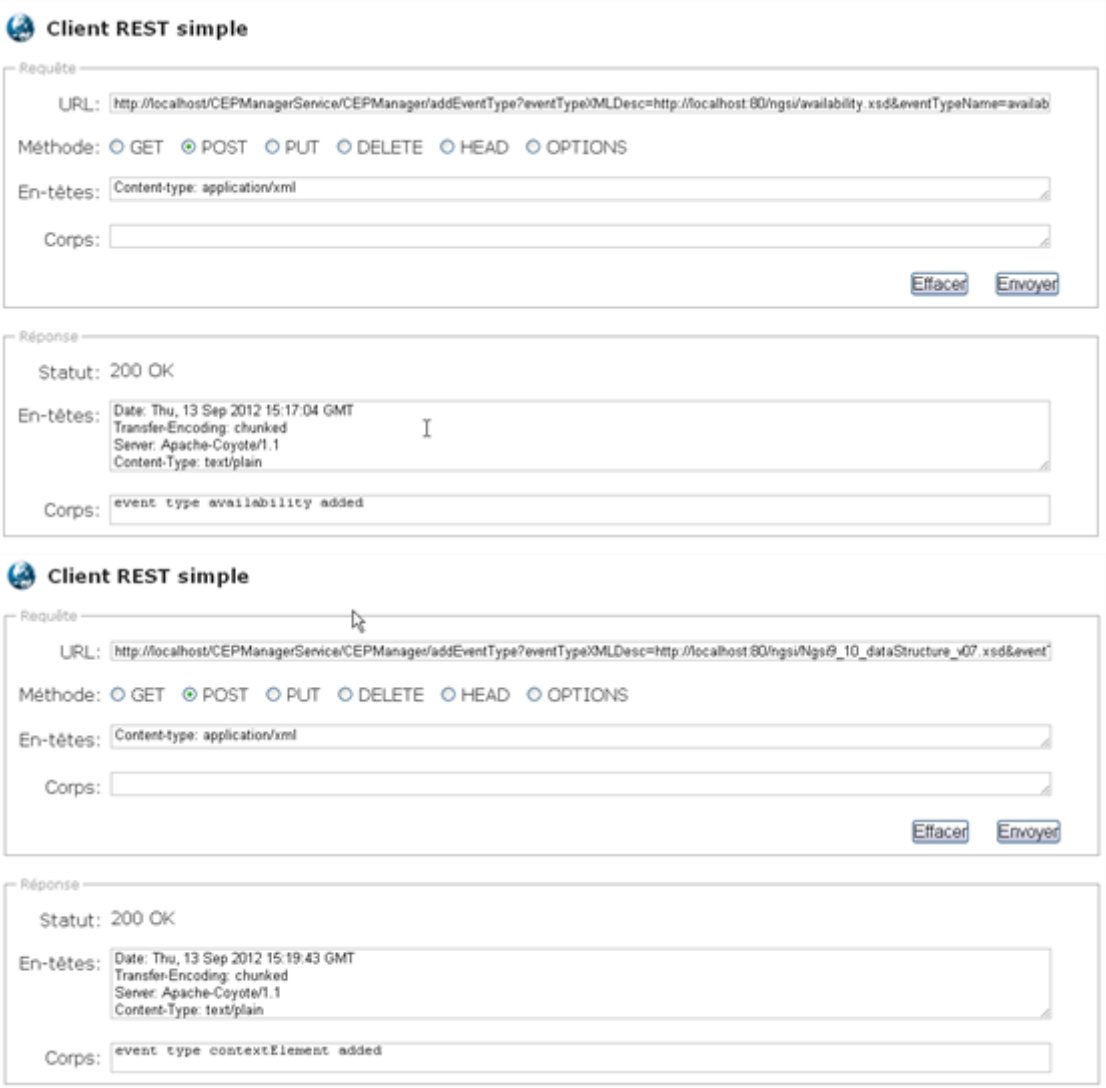
7. removeStatement POST request: remove a specific statement
URL:
http://localhost/CEPManagerService/CEPManager/removeStatement?statementName=stat0

8.2.1.3 *EventType methods*

1. getEventTypes GET request: get all the names of event types currently deployed in the CEP engine request:
URL: http://localhost/CEPManagerService/CEPManager/getEventTypes



2. addEventType POST request: add an event type to the CEP engine
URL:
http://localhost/CEPManagerService/CEPManager/addEventType?eventTypeXMLDesc=http://localhost:80/ngsi/availability.xsd&eventTypeName=availability&RootElement=availability
http://localhost/CEPManagerService/CEPManager/addEventType?eventTypeXMLDesc=http://localhost:80/ngsi/Ngsi9_10_dataStructure_v07.xsd&eventTypeName=contextElement&RootElement=contextElement



3. removeEventType POST request: remove a specific event type
URL:
http://localhost/CEPManagerService/CEPManager/removeEventType?eventTypeName=availability

4. sendEventFile POST request: send an event within the CEP
URL:
http://localhost/CEPManagerService/CEPManager/sendEventFile?XMLEventFile=http://localhost:80/ngsi/availability0.xml
http://localhost/CEPManagerService/CEPManager/sendEventFile?XMLEventFile=http://localhost:80/ngsi/contextElementEvent1.xml

4. sendEventFile POST request: send an event within the CEP
URL: http://localhost/CEPManagerService/CEPManager/sendEventXML

```
Payload :
<?xml version="1.0" encoding="UTF-8"?>
<pollution>
<sensorId>sensor1</sensorId>
<pollutionMessage>Mesure</pollutionMessage>
```

```
<pollutionType>SO2</pollutionType>
<pollutionLevel>1000</pollutionLevel>
<longitude>2.361650716814159</longitude>
<latitude>48.91714399001996</latitude>
<timeStamp>1333440195103</timeStamp>
</pollution>
```

8.2.2 Use Case from end to end

8.2.2.1 *Add Event Type*

Add Event Type availability, customerlocation, geolocation, stateorder and pollution
To add an event Type, use the google chrome Client REST simple with parameters:
- URL:
http://localhost/CEPManagerService/CEPManager/addEventType?eventTypeXMLDesc=http://localhost:80/ngsi/availability.xsd&eventTypeName=availability&RootElement=availability
- Method: POST
Event Type availability.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="availability" type="availability" />
```

```
  <xs:complexType name="availability">
    <xs:sequence>
      <xs:element name="userType" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="userId" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="status" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="phoneNumber" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="timeStamp" type="xs:time" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Event Type CustomerLocation.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="customerlocation" type="customerlocation" />
  <xs:complexType name="customerlocation">
    <xs:sequence>
      <xs:element name="customerId" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="address" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="latitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="longitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="phoneNumber" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="timeStamp" type="xs:time" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Event Type geolocation.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="geolocation" type="geolocation" />
  <xs:complexType name="geolocation">
    <xs:sequence>
      <xs:element name="userType" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="userId" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="latitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="longitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="phoneNumber" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="timeStamp" type="xs:time" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Event Type stateorder.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="stateorder" type="stateorder" />
  <xs:complexType name="stateorder">
    <xs:sequence>
      <xs:element name="customerPhoneNumber" type="xs:string"
minOccurs="1" maxOccurs="1" />
      <xs:element name="factoryId" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="status" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="timeStamp" type="xs:time" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Event Type pollution.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="pollution" type="pollution" />
  <xs:complexType name="pollution">
    <xs:sequence>
      <xs:element name="sensorId" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="pollutionMessage" type="xs:string"
minOccurs="1" maxOccurs="1" />
      <xs:element name="pollutionType" type="xs:string"
minOccurs="1" maxOccurs="1" />
      <xs:element name="pollutionLevel" type="xs:string"
minOccurs="1" maxOccurs="1" />
      <xs:element name="latitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="longitude" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="timeStamp" type="xs:time" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

8.2.2.2 *Add Statements*

Add statements statPollution and statAlert. To add a statement, use the google chrome Client REST simple with parameters:

- URL: http://localhost/CEPManagerService/CEPManager/addStatement?statementName=statPollution&statement=select+*+from+pollution
- Method: POST
  - statementName=statPollution, statement=select * from pollution

check statement state is STARTED

check Fields selected : sensorId, pollutionMessage, pollutionType, pollutionLevel, latidtude, longitude, timestamp

  - statementName=statAlert, statement=select * from pollution (pollutionMessage='Alerte')

8.2.2.3 *Add Action to Statements*

- statPollution: linked a REST getMethod, which open a file and write all params inside

- statAlert : linked a REST sendSms Method, which send an alert message to the administrator

JSON

```
{"statementName":"statPollution",
"restActionURI":"http://localhost/CEPManagerService/CEPManager/getMethod?sensorId=<id>&pollutionMessage=<msg>&pollutionType=<type>&pollutionLevel=<lev>",
"params":{"id":{"value":"sensorId","static":false},"lev":{"value":"pollutionLevel","static":false},"type":{"value":"pollutionType","static":false},"msg":{"value":"pollutionMessage","static":false}}}
```

JSON

```
{"statementName":"statAlert",
"restActionURI":"http://run.orangeapi.com/sms/sendSMS.xml?id=08f508b2714&from=20345&to=<to>&content=Alerte Pollution de <type> avec level=<level>","params":{"to":{"value":"33648737860","static":true},"level":{"value":"pollutionLevel","static":false},"type":{"value":"pollutionType","static":false}}}
```

8.2.2.4 *Send Events*

Send pollution events: Mesure or Alerte

To send events, use the google chrome Client REST simple with parameters:

- URL: http://localhost/CEPManagerService/CEPManager/sendEventXML
- Method: POST
- Payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<pollution>
<sensorId>sensor1</sensorId>
<pollutionMessage>Mesure</pollutionMessage>
<pollutionType>SO2</pollutionType>
<pollutionLevel>1000</pollutionLevel>
<longitude>2.361650716814159</longitude>
<latitude>48.91714399001996</latitude>
<timeStamp>1333440195103</timeStamp>
</pollution>
```

Alert event:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<pollution>
<sensorId>sensor1</sensorId>
<pollutionMessage>Alerte</pollutionMessage>
<pollutionType>SO2</pollutionType>
<pollutionLevel>1000</pollutionLevel>
<longitude>2.361650716814159</longitude>
<latitude>48.91714399001996</latitude>
<timeStamp>1333440195103</timeStamp>
</pollution>
```