Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.5.4.3: 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.5.4.3
Project Document Date: 2014-08-27
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium
1.1 Executive Summary

This document contains the User and Programmers guide of the IoT technical chapter in FIWARE. The purpose of this manual is twofold:

- Provide the users with insights in the functionality of each GE and how to use it
- Provide the developers with the right pointers to develop using the Open Specifications
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.3), 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.

From a physical standpoint, IoT enablers have been spread in two different domains:

- **FI-WARE IoT Gateway.** A hardware device hosting a number of features of one or several Gateway Generic Enablers of the IoT Service Enablement. It is usually located at proximity of the devices (sensors/actuators) to be connected. In the FI-WARE IoT model, the IoT Gateway is an optional element aiming to optimize the network traffic sent to the Backend and IoT services efficiency and reliability. Zero, one or more IoT Gateways can be part of a FI-WARE IoT setting. Several m2m technologies introduce specific gateway devices too, where it is not feasible to install FI-WARE gateway features. Those gateways are considered plain devices grouping other devices and not FI-WARE IoT Gateways.

- **FI-WARE IoT Backend.** A setting in the cloud hosting a number of features of one or several Generic Enablers of the IoT Service Enablement. It is typically part of a FI-WARE platform instance in a Datacenter. In the FI-WARE IoT model, a single IoT Backend is mandatory and it is connected to all IoT end devices either via IoT Gateway(s) and/or straight interfaces. Normally, during FI-WARE Releases R1 and R3 timeframes, the Backend will refer to the IoT Backend enablers installed in the FI-WARE Testbed or Open Innovation Lab (OIL), as described in the project Catalogue.

A key design statement is that, whenever present, IoT Gateways are not expected to be permanently connected to the Backend as per communications design or failures. Another relevant remark is that IoT Gateways are expected to be constrained devices in some scenarios. Therefore, light-weight implementations of the same GEs plus additional GEs...
interfaces helping to save unnecessary features/GEs are specially considered in the Gateway domain.

From the functionality point of view, FI-WARE IoT design aims to expose the "Things" abstraction to services developers, cope with different vertical m2m applications and provide a uniform access to heterogeneous m2m hardware and protocols. There is a number IoT features which are somehow duplicated in the Backend and the gateway domains in order to fulfill the goals and statements described above. For instance, a CEP engine at the Gateway level reduces the network overload and improves condition-based-events triggering time. Application developers will be able to access Things and devices observation and control interfaces in two ways:

- Directly, by using Northbound IoT interfaces as described in this Wiki.
- Throughout Data/Context GEs, by configuring Backend IoT GEs (IoT Broker) as NGSI notifications Context Providers of Data/Context Publish-Subscribe-Context-Broker GE.

Nota Bene: For the reader, we are using in the following chapters the same vocabulary as 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 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.
More information about the IoT Service Enablement Chapter and FI-WARE in general can be found within the following pages:

http://wiki.fi-ware.org

Internet_of_Things_Services_Enablement_Architecture

Materializing_Web_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.5.4.3_User_and_Programmers_Guide_front_page

Backend IoT Broker - IoT Broker - 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 our wikis. 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 part of the links contained in the deliverables generated from wiki pages cannot be rendered to fully working links. This happens for instance when a wiki page references a section within the same wiki page (but there are other cases). In such scenarios we preserve a link for readability purposes but this points to an explanatory page, not the original target page.

In such cases where you find links that do not actually point to the original location, we encourage you to visit the source pages to get all the source information in its original form. Most of the links are however correct and this impacts a small fraction of those in our deliverables.
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 those partners in their teams they have decided to involve.

1.8 Keyword list


1.9 Changes History

<table>
<thead>
<tr>
<th>Release</th>
<th>Major changes description</th>
<th>Date</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>First draft of deliverable submission generated</td>
<td>2014-07-22</td>
<td>TID</td>
</tr>
<tr>
<td>v2</td>
<td>First draft of deliverable submission generated</td>
<td>2014-08-27</td>
<td>TID</td>
</tr>
</tbody>
</table>
1.10 Table of Contents

1.1 Executive Summary ........................................................................................................... 2
1.2 About This Document .......................................................................................................... 3
1.3 Intended Audience .............................................................................................................. 3
1.4 Chapter Context .................................................................................................................. 3
1.5 Structure of this Document ................................................................................................ 5
1.6 Typographical Conventions ............................................................................................... 6
1.7 Acknowledgements ............................................................................................................. 7
1.8 Keyword list ....................................................................................................................... 7
1.9 Changes History .................................................................................................................. 7
1.10 Table of Contents ............................................................................................................. 8

2 Backend IoT Broker - IoT Broker - User and Programmers Guide ............................................. 10
    2.1 Introduction .................................................................................................................... 10
    2.2 User’s Guide ................................................................................................................... 10
    2.3 Developer Guide ............................................................................................................ 10

3 Configuration Manager - IoT Discovery - User and Programmers Guide ............................... 14
    3.1 Introduction .................................................................................................................... 14
    3.2 User’s Guide ................................................................................................................... 14
    3.3 Programmers Guide ....................................................................................................... 20

4 Backend Device Manager - IDAS - User and Programmers Guide ......................................... 33
    4.1 Introduction .................................................................................................................... 33
    4.2 User’s Guide ................................................................................................................... 33
    4.3 Programmer’s Guide ....................................................................................................... 33

5 Gateway Device Manager - OpenMTC - User and Programmers Guide ............................... 59
    5.1 Introduction .................................................................................................................... 59
    5.2 User’s Guide ................................................................................................................... 60
    5.3 Programmer’s Guide ....................................................................................................... 60

6 Gateway Protocol Adapter - ZPA - User and Programmers Guide ........................................... 64
    6.1 Introduction .................................................................................................................... 64
    6.2 User’s Guide ................................................................................................................... 65
    6.3 Programmers Guide ....................................................................................................... 66

7 Gateway Protocol Adapter - EPCGE - User and Programmers Guide ..................................... 86
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>86</td>
</tr>
<tr>
<td>7.2</td>
<td>User’s Guide</td>
<td>91</td>
</tr>
<tr>
<td>7.3</td>
<td>Programmer’s Guide</td>
<td>100</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>109</td>
</tr>
<tr>
<td>8.2</td>
<td>User’s Guide</td>
<td>109</td>
</tr>
<tr>
<td>8.3</td>
<td>Programmers Guide</td>
<td>110</td>
</tr>
<tr>
<td>9</td>
<td>Gateway Data Handling - Esper4FastData - User and Programmers Guide</td>
<td>111</td>
</tr>
<tr>
<td>9.2</td>
<td>User’s Guide</td>
<td>112</td>
</tr>
<tr>
<td>9.3</td>
<td>Programmer’s Guide</td>
<td>112</td>
</tr>
<tr>
<td>10</td>
<td>Template Handler - Template Handler - User and Programmers Guide</td>
<td>166</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>166</td>
</tr>
<tr>
<td>10.2</td>
<td>User’s guide</td>
<td>167</td>
</tr>
<tr>
<td>10.3</td>
<td>Programmer’s guide</td>
<td>173</td>
</tr>
</tbody>
</table>
2 Backend IoT Broker - IoT Broker - User and Programmers Guide

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

2.1 Introduction

Welcome the IoT Broker GE User and Programmer Guide. The IoT Broker is a backend component implemented by NEC in the IoT WP (WP5). It is built in JAVA over OSGi using standard interface NGSI 9/10 to communicate with the other components/GEs. The online documents are being continuously updated and improved, and the FI-WARE wiki will be the appropriate place to get the most up-to-date information on this GE.

2.1.1.1 Background and Detail

This User and Programmers Guide relates to the IoT Broker GE which is a backend component part of the Internet of Things (IoT) Services Enablement Architecture. Please find more information about this Generic Enabler in the following OpenSpecification for IoTBroker.

2.2 User’s Guide

The Backend IoT Broker is a backend enabler therefore there isn’t need to provide a user guide but a programmer guide, which is provided in the next section.

The NGSI-10 reference (OMA_NGSI-10) describes how to use the NGSI-10 API in detail. For using the IoT Broker you need to contact via HTTP the server on port 80 with one of the REST HTTP METHOD (GET, POST, PUT, DELETE) according to the NGSI-10 reference document.

In addition to the basic OMA NGSI 10 interface, the NEC IoT Broker also implements full support of the NGSI entity-to-entity association concept; please follow this link for documentation.

In the following section it is detailed how to use the IoT Broker GE from a user or developer perspective.

2.3 Developer Guide

The IoT Broker exposes an NGSI-10 interface, which is a RESTful interface over HTTP. This means that it is possible query the IoT Broker regardless of the programming language.

First, for checking if the IoT Broker GE is running and which operations are supported, is possible to send a GET on http://[IoT Broker IP]/. The response will be like this:
What is shown is the dashboard of the running IoT Broker instance. From this dashboard, the set of supported operations can be verified by clicking on the "Operations" tab.

Now for getting a feeling of how NGSI-10 works, let us send a GET request on http://[IoT Broker IP]/ngsi10/contextEntities/Kitchen. The HTTP request and response headers are showed below:

```
<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
```
It is important to note that the NGSI-10 API, exposed by IoT Broker GE, required as Content-Type "application/xml", which means that an application can send only xml content and receive only xml as content.

2.3.1 Accessing the IoT Broker NGSI-10 Interface from a Browser
The following example interactions can be executed using the Chrome browser [1] with the 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. Further examples of NGSI 10 usage can be found in the NGSI documentation.

1. GET request:
2. POST request:

The request message body is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
    <entityIdList>
        <entityId type="Room">
            <id>Kitchen</id>
        </entityId>
    </entityIdList>
    <attributeList>
        <attribute>indoorTemperature</attribute>
    </attributeList>
</queryContextRequest>
```
3 Configuration Manager - IoT Discovery - User and Programmers Guide

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

3.1 Introduction

This guide will explain how to interact with the IoT Discovery GEi and also the APIs that are exposed by it. The GEi itself is made of two components. The first being the Sense2Web Platform and the NGSI-9 Server module.

3.1.1 Background and Detail

This User and Programmers Guide relates to the Configuration Management GE which is part of the Internet of Things (IoT) Services Enablement chapter. Please find more information about this Generic Enabler in the following Open Specification.

3.2 User’s Guide

3.2.1 NGSI-9 Server

The NGSI-9 Server is a server with a backend repository that allows the registration and discovery of NGSI-9 Context Entities. Interaction with the server can be done via a REST web service client, or a web browser using the a REST Client Plugin, e.g. Google Chrome with Advanced REST Client.
3.2.2Sense2Web Platform

Sense2Web is a platform for supporting semantic linked-open-data (LOD) descriptions that model IoT Resources, Entities and Services, which are based on the IoT-A ontologies. It provides a set of triple stores for the storage of these descriptions, and exposes a Web User Interface (UI) and RESTful interfaces for CRUD (Create/Read/Update/Delete) management. In turn, the platform provides a set of search mechanisms for their lookup and discovery, namely the Probabilistic Search Engine and the Associations Engine. Please refer to the IoT-A ontologies can be found in via this link.

3.2.2.1Web User Interface

3.2.2.1.1Home page

The Web UI can be accessed via http://{serverRoot}/S2W. The main page displays page links to the current features that the platform currently supports. Click on the associated icon for a particular feature.
3.2.2.1.2 **Register a Description**

This page provides the user to register an IoT description. The current ontologies that are supported are the IoT-A ontologies that define a **Resource**, **Virtual Entity** and **Service**. Please refer to this [paper](#) for more details on the IoT-A ontologies. Registering can be done either by uploading a description file or by completing a form. The form is provided for entering values for the description properties. The form will not be accepted unless an ID, Name and Latitude/Longitude co-ordinates are at least entered. The tag field is used to provide a keyword for the description. The linked-data tag is used to link the description with another description, which can be used to provide more detailed information about a particular element in the description. Currently linked-data tags can only be retrieved from “dbpedia.com”. To retrieve a linked-data tag the user must enter a keyword. Then double-clicking the text box will trigger the retrieval of a set of linked-data tag to choose from. The search results can be refined by specifying a topic and also by limiting the number of results. The local and global location fields also provide linked-data descriptions that can be associated with a particular location. Location can also be entered in terms of co-ordinates and altitude. A Google mini-map can be used to assist in entering the co-ordinates. To select a point in the map, just simply click on the point required, or alternatively the marker can be dragged to the point required. Once the form is submitted the page will return links for viewing the RDF result of the submission in various formats i.e. **RDF/XML**, **RDF/XML-ABBREV**, **RDF/JSON**, **N-TRIPLE** and **TURTLE**. If the ID entered for this description is the same as one already available in the repository, then an error message is returned.
3.2.2.1.3 Lookup a Description

To lookup a description, the relevant repository needs to be selected i.e. Resource, Entity, Service. The ID of the description in question must then be entered.

3.2.2.1.4 Update a Description

The Update page is similar to the Register page with the exception that the current values for a description can be retrieved and populated into the fields by entering the ID of the description in question, and clicking on the "retrieve" button.
3.2.2.1.5 **Delete a Description**

To delete a description, the relevant repository needs to be selected i.e. Resource, Entity, Service. The ID of the description in question must then be entered.

3.2.2.1.6 **Query a Description**

The platform supports SPARQL for querying IoT descriptions. When choosing a particular type of description, the respective SPARQL template is provided for a user to use and edit. The template includes the properties of a description type. In the case of finding associations between Entities and Services, the OPTION field in the SPARQL template for the Entity can be used to retrieve IoT Service Description URI, which can then be used by a user/application to retrieve information on how to reach a Service Endpoint that provides information that is currently relevant to the attributes of an Entity e.g. a Service of a temperature sensor (i.e. the
Resource) attached to a mobile target that is currently in the vicinity of a room (i.e. the Entity in question).

3.2.2.1.7 Discover a Description

In the case where a user does not know the description or the exact naming for its attributes that the user is looking for, the probabilistic search engine can be used to provide recommended and ranked suggestion for a description relevant to the search input. Here the user should enter a keyword for as many fields as required.

3.2.2.1.8 Locate a Description

A simple map application is provided to show the location of a Resource or Entity. Clicking on a particular Object will display its main properties and a link to its description.
3.3 Programmers Guide

The GEi is meant to be used as a web service, and therefore users and applications can interact with the GEi via two sets of RESTful interfaces.

3.3.1 API overview

The subsections below give an overview of the RESTful API for the NGSI-9 Server and the Sense2Web Platform.

3.3.1.1 NGSI-9 API

Please refer to the FI-WARE NGSI-9 Open RESTful API Specification on the details on the API, and also the NGSI associations concept for details on how to register and discover associations.

The Standard Operations currently supported are:

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><code>://{serverRoot}/ngsi9/registerContext</code></td>
<td><code>registerContextRequest</code></td>
</tr>
<tr>
<td>POST</td>
<td><code>://{serverRoot}/ngsi9/discoverContextAvailability</code></td>
<td><code>discoverContextAvailabilityRequest</code></td>
</tr>
<tr>
<td>POST</td>
<td><code>://{serverRoot}/ngsi9/subscribeContextAvailability</code></td>
<td><code>subscribeContextAvailabilityRequest</code></td>
</tr>
<tr>
<td>POST</td>
<td><code>://{serverRoot}/ngsi9/updateContextAvailabilityS</code></td>
<td><code>updateContextAvailabilitySubscriptionRequest</code></td>
</tr>
</tbody>
</table>
The Convenience Operations currently supported are:

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>//{serverRoot}/ngsi9/contextEntities/{EntityID}</td>
<td>registerContextRequest</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/ngsi9/contextEntities/{EntityID}</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/ngsi9/contextEntities/{EntityID}/attributes</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/ngsi9/contextEntities/{EntityID}/attributes/{attributeName}</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/ngsi9/contextEntities/{EntityID}/attributeDomains/{attributeDomainName}</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>//{serverRoot}/ngsi9/contextEntityTypes/{typeName}</td>
<td>N/A</td>
</tr>
</tbody>
</table>

3.3.1.1 "Hello World" Example

Registration of Context Entities

The first step is to Register a Context Entity. So, using a REST client, we will send an NGSI registerContextRequest message like the one shown below using the following URL:

POST http://{serverRoot}/ngsi9/registerContext

The payload as an example can be the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
  <contextRegistrationList>
    <contextRegistration>
      <entityIdList>
        <entityId type="Room" isPattern="false">
```
<id>ConferenceRoom</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>
        </metadata>
    </contextRegistrationAttribute>
</contextRegistrationAttributeList>
<registrationMetaData>
    <contextMetaData>
        <name>ID</name>
        <type>string</type>
        <value>2212</value>
    </contextMetaData>
</registrationMetaData>

<providingApplication>http://192.168.100.1:70/application</providingApplication>

Results obtained: should be a registerContextResponse similar to the following, whereby the registration number will be randomly generated.

<?xml version="1.0" encoding="UTF-8"?>
<registerContextResponse>
    <duration>PT1M</duration>
    <registrationId>UNIS887313</registrationId>
    <errorCode>
        <code>200</code>
        <reasonPhrase>OK</reasonPhrase>
        <details xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:type="xs:string">Stored</details>
    </errorCode>
</registerContextResponse>
Discovery of Context Entities

The next step is to discover the availability of a Context Entity. So, using a REST client, we will send a discoverContextAvailabilityRequest message like the one shown below:

```xml
POST http://{serverRoot}/ngsi9/discoverContextAvailability

<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityRequest>
  <entityIdList>
    <entityId type="Room" isPattern="false">
      <id>ConferenceRoom</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>
      <operationScope>
        <scopeType></scopeType>
        <scopeValue></scopeValue>
      </operationScope>
    </scope>
  </restriction>
</discoverContextAvailabilityRequest>
```
The response should be of the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<discoverContextAvailabilityResponse>
  <contextRegistrationResponseList>
    <contextRegistrationResponse>
      <contextRegistration>
        <entityIdList>
          <entityId type="Room" isPattern="false">
            <id>ConferenceRoom</id>
          </entityId>
        </entityIdList>
        <contextRegistrationAttributeList>
          <contextRegistrationAttribute>
            <name>temperature</name>
            <type>degree</type>
            <isDomain>false</isDomain>
            <metadata>
              <contextMetadata>
                ...
              </contextMetadata>
            </metadata>
          </contextRegistrationAttribute>
        </contextRegistrationAttributeList>
      </contextRegistration>
    </contextRegistrationResponse>
  </contextRegistrationResponseList>
</discoverContextAvailabilityResponse>
```
<value></value>
</contextMetadata>
</registrationMetaData>

<providingApplication>http://192.168.100.1:70/application</providingApplication>
</contextRegistration>
</contextRegistrationResponse>
</contextRegistrationResponseList>
<errorCode>
  <code>200</code>
  <reasonPhrase>Ok</reasonPhrase>
  <details>a</details>
</errorCode>
</discoverContextAvailabilityResponse>
### 3.3.1.2 Sense2Web API

The diagram below illustrates the structure of RESTful API:

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>//{serverRoot}/S2W/repository/getVersion</td>
<td>getVersion: returns the current version of the GEi</td>
</tr>
<tr>
<td>POST</td>
<td>//{serverRoot}/S2W/repository/register/iot-a/{descriptionType}</td>
<td>registerMultipleDescriptions: registers a set of descriptions with the S2W platform. {descriptionType} can be &quot;resource&quot;, &quot;entity&quot; or &quot;service&quot;. The descriptions that are submitted is must</td>
</tr>
</tbody>
</table>
POST  
\[//\{serverRoot\}/S2W/repository/register/iot-a/\{descriptionType\}/\{objectID\}\]

**registerDescription**: registers a description with the S2W platform. 
{descriptionType} can be "resource","entity" or "service". The description that is submitted must conform with the IoT-A ontologies.

GET  
\[//\{serverRoot\}/S2W/repository/lookup/iot-a/\{descriptionType\}/\{objectID\}?resultFormat={resultFormat}\]

**lookupDescription**: looks up a description with a specific ID and retrieves it in a specific format: "RDF/XML", "RDF/JSON", "N3", "N-TRIPLE", "TURTLE".

PUT  
\[//\{serverRoot\}/S2W/repository/update/iot-a/\{descriptionType\}/\{objectID\}\]

**updateDescription**: updates a description with a specific ID that is already registered with the platform's repository.

DELETE  
\[//\{serverRoot\}/S2W/repository/delete/iot-a/\{descriptionType\}/\{objectID\}\]

**deleteDescription**: deletes a description with a specific ID that is already registered with the platform's repository.

GET  
\[//\{serverRoot\}/S2W/repository/query/sparql/iot-a/\{descriptionType\}?sparql={sparqlQuery}\?resultFormat={resultFormat}\]

**queryDescription**: query the repository using SPARQL. The SPARQL query must be inserted in the "sparql" query parameter, with the query itself being URL-encoded for it to be sent as a GET request. "resultFormat" can be "XML", "JSON", "CSV", "TSV", or "BIO".

POST  
\[//\{serverRoot\}/S2W/repository/query/sparql/iot-a/\{descriptionType\}?resultFormat={resultFormat}\]

**queryDescription**: query the repository using SPARQL. The SPARQL query is inserted in the POST payload. "resultFormat" can be "XML", "JSON", "CSV", "TSV", "BIO".

POST  
\[//\{serverRoot\}/S2W/repository/discover/iot-a/\{descriptionType\}\]

**discoverDescription**: discover a description by submitting a template of a particular description type, which contain values as keywords for the search.
3.3.1.2 "Hello World" example

In this example, we will register an IoT-A description at the Sense2Web repository using the HTTP method and URL below:

```
POST - http://{serverRoot}/S2W/repository/register/iot-a/resource/Resource_16_BA_02_temperature_sensor
```

We will then retrieve the same description using the HTTP method and URL below:

```
GET - http://{serverRoot}/S2W/repository/lookup/iot-a/resource/Resource_16_BA_02_temperature_sensor
```

The POST method will contain in its payload a description that conforms to the IoT-A ontology, such as the one below:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns="http://www.surrey.ac.uk/ccsr/ontologies/ResourceModel.owl#"
   xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dm="http://www.surrey.ac.uk/ccsr/ontologies/DeviceModel.owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:swrl="http://www.w3.org/2003/11/swrl#"
   xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

<OnDeviceResource rdf:ID="Resource_16_BA_02_temperature_sensor">
  <hasResourceLocation>
    <Location rdf:ID="Location_16_BA_02_temperature_sensor">
      <hasLongitude rdf:datatype="http://www.w3.org/2001/XMLSchema#float">-0.57</hasLongitude>
      <hasLocalLocation rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.surrey.ac.uk/ccsr/ontologies/LocationModel.owl#E16</hasLocalLocation>
      <hasGlobalLocation rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.geonames.org/2647793/</hasGlobalLocation>
      <hasAltitude rdf:datatype="http://www.w3.org/2001/XMLSchema#int">45</hasAltitude>
      <hasLatitude rdf:datatype="http://www.w3.org/2001/XMLSchema#float">51.23</hasLatitude>
    </Location>
  </hasResourceLocation>
  <isHostedOn><j.0:SensingDevice rdf:ID="PloggBoard_16_BA_02_temperature"/></isHostedOn>
  <hasResourceID rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">16_BA_02_temperature_sensor</hasResourceID>
  <hasType rdf:resource="#Sensor"/>
  <isExposedThroughService rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
The response, which reports the result of the registration, should be a JSON response with the following format:

```
{
  id: "Resource_16_BA_02_temperature_sensor",
  type: "resource",
  stored: true,
  indexed: true,
  association: ""
}
```

- the GET request should return the same description as the one registered with the POST example.

Please refer to the Unit Test Plan wiki page for this GEi to get examples on how to interact with the web server using IoTA models and SPARQL queries.
4 Backend Device Manager - IDAS - User and Programmers Guide

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

4.1 Introduction


IDAS is fully a backend asset that is able to collect and store events from physical devices with or without the presence of intermediate gateways. It is also capable of forwarding commands to bidirectional devices (actuators).

4.1.1.1 Background and Detail

This User and Programmers Guide relates to the Backend Device Management GE, which is part of the Internet of Things (IoT) Services Enablement Architecture. Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.IoT.Backend.DeviceManagement.

4.2 User's Guide

IDAS users are always M2M application developers and therefore this manual has no user's guide section but only the programmer's one.

4.3 Programmer's Guide

From the software components point of view, IDAS is made of two assets: the IDAS M2M platform and the IoT-Agent, which plays the role of NGSI connector to external capable components. IDAS platform consists in turn in three sub-components: "application services", "real-time event processing" and "Device Communication Handling".

The following picture shows the complete component architecture of IDAS BE Device Management GEi.
As depicted above, three communication pipes with their respective APIs are exposed:

- **ADMINISTRATION REST API**: This is the main API offered to programmers. It offers services to M2M-aware applications for creating an M2M service framework, subscribe to events of devices within an M2M service, send commands to bidirectional devices and perform pre-defined queries to get history information of devices. The API has been designed and implemented by IDAS team keeping in mind simplicity and usability as design criteria.

- **DEVICE COMMUNICATION REST API**: This is the primary API exposed to physical devices and sensors and it is based on SensorML standard using JSON codification. In order to optimize performance for constrained environments, light and ultra-light standards are supported in addition to SensorML. IDAS FI-WARE team is working on adding ETSI M2M mId support in addition as part of this GEi roadmap (Release 3.1).

- **NGSI9/10 REST API**: This is the API offered by the IoT-Agent connector. Programmers can this way configure IDAS to play the role of an NGSI Context Provider so that other external NGSI-enabled components (many in thr FI-WARE Catalogue) are able to receive physical devices events coded as NGSI notifications.

The following sections explain in detail each one of the above described IDAS REST APIs.
4.3.1 Administration REST API

This API is exposed by the IDAS M2M platform, concretely by the "Application Services" subcomponent. The operations included in this API allow these applications:

- To subscribe to receive events from devices
- To be notified when an event arrives triggering a subscription
- To send commands to a device and to receive the results from them
- To obtain and provision data from/to the underlying M2M network
- To provision customer information

4.3.1.1 Security Considerations

From the security standpoint, all communication with this Rest API is performed over http. Any security feature should be provided by an external platform.

4.3.1.2 Data Format & Model

Regarding Data format, all API requests can use XML or JSON format. IDAS organizes data into a hierarchy of different components. Top Level object is service and represents the client that owns all the others components.

For the organization of resources the following basic concepts are relevant:

- **Service (M2M service framework)**: the owner of assets/devices.
- **Group**: logical grouping of assets/devices.
- **Asset/Device**: Electronic or virtual component that provides data to the M2M platform.
- **Model**: Group of devices with same capabilities. Users can define models for provisioning devices with the same measurement capabilities. In this case, devices don’t need to send register message because their model describes them. A model is defined belonging to a service.

4.3.1.3 Unified Resources Summary

The following picture shows the resource tree structure implemented by the IDAS ADMINISTRATION REST API. Note that those nodes in the resource tree which have associated HTTP methods defined in this specification are depicted by solid boxes.
Where:

- `{apiRoot}` is defined as: `{serverRoot}/DCA/{apiVersion}
- `{apiVersion}` is the API version in this case v2
- and `{serverRoot}` is implementation specific (e.g. api.telefonica.com).

### 4.3.1.4 API Operations

The functionality provided by this API is separated in five blocks. Every block has several entities, every entity has a specific and unique URI, several operations (GET, POST, PUT, DELETE) but not all entities have all the operations. The following subsections show a table per each functionality block with its admitted operations over resources.

A full description of the operations, parameters, expected results and examples is available in the [IDAS ADMIN REST API manual](#) (Please, note that in this Telefonica manuals we refer to IDAS GEi as IDAS, DCA or IDAS-DCA).

#### 4.3.1.4.1 Provision

This functional block is used to configure internal components. Operations are described in the next subsection "Data Model".

#### 4.3.1.4.2 Data Model

This block covers the configuration of IDAS devices hierarchy (Service, Models, Groups and Asset/Device).
4.3.1.4.3  Sensor Data

This functional block is used in order to retrieve the last measure and historical measures published by a device.

4.3.1.4.4  Command

Devices can receive commands in order to make some internal operations. This functional block describes how an application can send a command to a device. Commands are XML documents described in IDAS Gateway interface (see "DEVICE COMMUNICATION REST API"
4.3.1.4.5 Subscription

This functional block describes the procedures for an application to receive information published by a device.

4.3.2 Device Communication REST API

This API is exposed by the IDAS M2M platform, specifically by the "Device Communication Handling" subcomponent. This subcomponent hosts different agents or plug-ins implementing different protocols to talk to various kinds of physical devices. Currently, "SOHttpService" and "SEHttpCommands" are implemented, while "ETSIM2MHtttpService" will be released in the future, as part of FI-WARE R3.1.

The "Device Communication Handling" is also called sometimes as the "IDAS Gateway" as long as it performs some of the tasks of M2M gateways at the backend side. This enables practical M2M scenarios where sensors or actuators can be directly connected to IDAS without the need of physical gateways as intermediate elements. However, nothing prevents IDAS to work with physical gateways that are able to gather events from multiple physical devices and forward them coded in SensorML to IDAS.

The operations including the description of an action or measure taken by a resource consist in XML documents transmitted through the HTTP 1.1 protocol in the body of a POST request.
It is mandatory for any physical device/asset to send a registration request before sending any measure. A sending of a resource measure not previously registered will result in an error.

The following section show the operations exposed by this API, their parameters and examples. A full description of the API operations, data model, SensorML codification and Lightweight alternative is described in the IDAS DEVICE COMMUNICATION API manual (Please, note that in this Telefonica manuals we refer to IDAS GEi as IDAS, DCA or IDAS-DCA).

### 4.3.2.1 Registration Request

It is used to describe resources and register them on the platform. Registration can be done both with the traditional protocol (SensorML), and with the lightweight protocol described in the full API description.

```xml
<sos:RegisterSensor service="SOS" version="1.0.0"
   xsi:schemaLocation="http://www.opengis.net/sos/1.0 sosRegisterSensor.xsd"
   xmlns:om="http://www.opengis.net/om/1.0"
   xmlns:swe="http://www.opengis.net/swe/1.0.1"
   xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
   xmlns:sos="http://www.opengis.net/sos/1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  
  <sos:SensorDescription>
    
    <sml:System/> ver REF _Ref258836308 \r \h 3.1
    
    08D0C9EA79F9BACE118C8200AA04BA90B0200000080000000E0000005F0052
    00650066003200350038003800330030003800000
    
    </sos:SensorDescription>
  
</sos:ObservationTemplate>
```
This XML document is sent in the Content of a HTTP 1.1 POST request.

The response to a successful petition is an XML document in the Content of an HTTP 1.1 200 response.
<sos:RegisterSensorResponse
xsi:schemaLocation="http://www.opengis.net/sos/1.0
sosRegisterSensor.xsd"
xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<sos:AssignedSensorId>WeatherStation_1</sos:AssignedSensorId>

</sos:RegisterSensorResponse>

The `sos:AssignedSensorId` identifier is of the resource registered as System.

In case of the lightweight protocol, a successful registration response of a resource is:

```xml
<rsr id="Resource Identifier"/>
```

When an error occurs, a HTTP 1.1 4xx code is returned with Content:

```xml
<ows:ExceptionReport

version="1.0.0"
```
In case the lightweight protocol is used, an error condition is described as:

```
<e c="código" l="locator">Error code description</e>
```

### 4.3.2.1.1 Types of Registration

Registration can proceed from two sources, from a device, this is the registration procedure above described and can also proceed from the API REST.

In the first case, when the register message arrive from a device can include a parameter modelName or cannot have this parameter. The gateway sends the register message to the sensor enabler and this platform manages the model. If the parameter modelName is present, and the model exists in database, the registration is according the existing model. If the model not exists in database, the sensor enabler generates a model with the information of the
register message, and the name of the modelName. When the register has not the parameter modelName, the sensor enabler generates a model in database with the information of the register message, and the name “Autogenerate”.

In the second case, the gateway receives the registration from the API REST, through neo_http_proto. In this case, registration is always according with a model. The message received from the API REST includes the following information:

```plaintext
model_name : name of model
service : number of service
domain : name of concentrator
device : name of device
enabled_command : enable or disable command functionality
lon, lat : for location
```

The gateway find in the mongo collection MODEL, the information for registration associated to model_name and service. Using the input / output information that read from the database, the gateway generates a register message, and sends it to the neo_oe_recorder process in the sensor enabler.

An example of the information in database about the model is:
"MN" : "ModelJUL",

"IL" : [
    {
        "NI" : "Temperatura",
        "OB" : "temperature",
        "O" : {
            "NO" : "Temperatura",
            "AUL" : "Temperatura",
            "T" : "Quantity",
            "UOM" : "partsPerMillion"
        }
    }
]
"NI" : "Humedad",

"OB" : "relativeHumidity",

"O" : {

"NO" : "Humedad",

"AUL" : "h",

"T" : "Quantity",

"UOM" : "percent"
}

"SERVICE" : 2402

Where IL indicates the list and format of inputs and outputs allowed for the device.

If the field AUL is dummy or not exist, that indicates a register in SensorML protocol, if AUL is not dummy indicate register in Ligth protocol.
4.3.2.2  Measure Notification Request

This request is used to inform the platform about the collection of an observation or measurement. You can perform both SensorAPI or lightweight protocol described in the full API description.

```xml
<sos:InsertObservation service="SOS" version="1.0.0"
xsi:schemaLocation="http://www.opengis.net/sos/1.0
sosInsert.xsd" xmlns:om="http://www.opengis.net/om/1.0"
xmlns:sos="http://www.opengis.net/sos/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<sos:AssignedSensorId>WeatherStation_1</sos:AssignedSensorId>

<om:Observation>

<om:samplingTime/>

<om:procedure/>

<om:observedProperty/>

<om:featureOfInterest/>

<om:result/>

</om:Observation>

</sos:InsertObservation>
```
This document is sent in the Content of a HTTP 1.1 POST request.

The response to a request with success is an XML document in the Content of a HTTP 1.1 200 response.

```xml
<sos:InsertObservationResponse
 xsi:schemaLocation="http://www.opengis.net/sos/1.0
 sosInsert.xsd" xmlns:sos="http://www.opengis.net/sos/1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<sos:AssignedObservationId>1</sos:AssignedObservationId>

</sos:InsertObservationResponse>
```

This returns an identifier for the inserted observation.

If we use lightweight protocol to publish measure, the successful response is:

```xml
<ior id="Measure Identifier"/>
```

In case of error it returns a HTTP 1.1 4xx code as shown in 6.1. The response with error to the publication of an observation is similar to the described for the case of resource record through the lightweight protocol.

Using the super-small protocol we can send multiple frames for several observations. Each observation will be separated by a configurable character.

The super-small protocol allows specifying certain parameters in the URL:
- UC: indicates a UniversalIdentifierOfLogicalHub. Its presence allows not transmitting this data for each observation included in the HTTP POST. This is useful when several observations are transmitted from the same UC.

- ID: Indicates a resource. Its presence allows not transmitting this data for each observation included in the HTTP POST. This is useful when several observations are transmitted from the same resource.

- URL: Indicates the address to which we send commands. This parameter must be accompanied by the two previous ones, and allows updating the direction in which the commands are received by the resource indicated by ID, and the hub indicated by UC.

4.3.2.3 **Commands and Actions**
The mechanism for running commands or actions in the environment of the network of sensors and actuators is similar.

It is essential to know the subject(s) to be changed. The case of actuators is reduced to the modification of parameters that have an effect on some physical phenomenon.

The commands are sent to the switch via HTTP POST method to the URL specified in the parameter defined as urn:x-ogc:def:property:IDAS:1.0:commandURL in the RegisterSensor request. The message content will be:

```xml
<paid:command dest="" name="" id="">
    <paid:cmdParam>
        Tipo definido en RegisterSensor para el parámetro.
    </paid:cmdParam>
</paid:command>
```
The name attribute indicates the command string to execute, and using the id attribute assigns a sequence number to the command. The dest attribute specifies the resource recipient of the command.

The parameters are referenced using the URN with that it was described in the RegisterSensor message parameter.

### 4.3.2.3.1 SET/GET default Commands

By default, any parameter posted in the RegisterSensor message is accessible through the SET/GET command. When a parameter is set to read only the GET command applies. When a write command is defined only applies the SET command. When a parameter is defined readwrite, both commands apply.

- **GET command**: should be applied to parameters defined or described as read and allows recovery of the value of that attribute. The name attribute is GET.

- **SET command**: should apply to parameters defined as writing and including the value with an attribute to be updated. With this type of command you can change settings or perform actions. The name attribute is SET.

Let’s suppose that a parameter in the RegisterSensor was defined as:

```xml
  <swe:Quantity definition="urn:x-ogc:def:phenomenon:IDAS:1.0:time">
    <swe:uom xlink:href="urn:x-ogc:def:uom:IDAS:1.0:millisecond"/>
    <swe:value>30000</swe:value>
  </swe:Quantity>
</sml:parameter>
```
Note that the parameter is defined through a urn namespace outside the IDAS one (XXX), since, in principle, has in itself no special semantics in the namespace of the platform.

To read the parameter value is send a GET command:

```xml
<paid:command
 xsi:schemaLocation="urn:ogc:def:dictionary:PAID:1.0:paid
 PaidCommand.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:swe="http://www.opengis.net/swe/1.0.1"
 xmlns:paid="urn:ogc:def:dictionary:PAID:1.0:paid"
 dest="id_recurso" name="GET" id="1234">
 <paid:cmdParam name="urn:x-ogc:def:property:XXX:1.0:TemperaturePeriod"/>
</paid:command>
```

To set the value of this parameter will be sent a SET command:

```xml
<paid:command
 xsi:schemaLocation="urn:ogc:def:dictionary:PAID:1.0:paid
 PaidCommand.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:swe="http://www.opengis.net/swe/1.0.1"
 xmlns:paid="urn:ogc:def:dictionary:PAID:1.0:paid"
 dest="id_recurso" name="SET" id="1234">
 <paid:cmdParam name="urn:x-ogc:def:property:XXX:1.0:TemperaturePeriod"/>
</paid:command>
```
4.3.2.3.2  Specific Commands

It is possible to describe specific commands via parameters defined as urn:x-ogc:def:property:IDAS:1.0:commandProperty in the xlink:role attribute.

For example, to define a Reset command (without parameters), a parameter in RegisterSensor is defined as:

```
</sml:parameter>
```

If there were parameters, we would define a swe:DataRecord and each field would be a parameter.
And the command to send would be:

```xml
<paid:command
 xsi:schemaLocation="urn:ogc:def:dictionary:PAID:1.0:paid
 PaidCommand.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:swe="http://www.opengis.net/swe/1.0.1"
 xmlns:paid="urn:ogc:def:dictionary:PAID:1.0:paid"
 dest="urn:ogc:def:dictionary:PAID:1.0:paid"
 id="id_recurso" name="urn:x-ogc:def:property:XXX:1.0:Reset " id="1234">
```
4.3.2.3 Commands Responses

In many situations, the request for execution of a command is not performed instantaneously by agencies with the protocol used in the sensor network. There are networks in which the nodes follow sleep/awake procedures and a command attention depends on the time they are active or awake.

Thus, the command responses follow two mechanisms:

- **Synchronous**: When the command is executed immediately. In this case, a HTTP 200 OK code will be sent when the command is successful, or appropriate HTTP code to indicate an error. It may incorporate a response message body.

- **Asynchronous**: when the execution is delayed. In this case, it shall be replied with a HTTP error code in the event that the command can not be executed, or a HTTP 202 Accepted code if the request is executed on a deferred basis. After running the command we may proceed to send the result of running through a HTTP POST command and the result of execution.

A response to a command follows the following scheme:
There should always be the id attribute to identify the command that is answered and that correspond to the id attribute of the command execution request.

When it comes to the default SET/GET commands affecting a parameter, it will return the parameter value in the format corresponding to that parameter (paid:cmdParam).

When it comes to a specific command, the answer will be paid:resultCommand type, containing one of the basic types defined in the interface (e.g. swe:Boolean).

When we want to explain an error, it will be replied with paid:errorCommand, which is a string type.

If a command is executed synchronously and the request is correct, it always returns a 200 OK, even if the result of the execution contains an errorCommand. That is, the HTTP code identifies whether the request is correct and commandResponse determines the result of command execution.

The reduced protocol associated with sending commands is the use of the types defined for that protocol.

The following table indicates the label replacement towards the reduced protocol:

4.3.2.3.4 Commands by Polling from Devices

Alternative mechanism is provided in order to execute a command. When a device publishes its description by RegisterSensor operation without commandURL parameter, platform cannot send a command by HTTP POST method.

In this scenario, command is temporarily stored waiting HTTP GET method from device to retrieve it:
• HTTP method: GET
• Device uses the URL to send measures with parameters to identify itself:

   GET /idas/sml?UC=<UniversalConcentrator>&ID=<device>

• Body response is a command as described above.
• When command is executed, device informs as asynchronous execution.

4.3.2.3.5 Updating Command URL of Device
When command URL of device changes, the new URL must be informed. The mechanism consists of sending HTTP POST method with parameters (this parameters can be sent when a measure is sent). Parameters:

• UC: identifies UniversalConcentrator.
• ID: identifies the device or resource changing command URL.
• URL: identifies the new command URL.

4.3.2.3.6 The API-KEY
As another level of security that enables client side authentication, the secret concept is introduced.

In the service provisioning phase, there is the option of creating an API KEY. If this option is chosen, all the requests related to this service must contain “apikey=[generated key]”.

This key will be provided to the devices so they can include it into the URL. It allows indentifying the source of the requests to some extent.

4.3.3 NGSI9/10 REST API
This API is exposed by the IoT-Agent connector component. Once an M2M service framework has been created and sensors are notifying events to IDAS M2M platform, this connector enables external NGSI applications to discover entities providing context updates via NGSI9 operations and to subscribe to those notifications via NGSI10 operations.

Because of this, the NGSI REST API of the IDAS (concretely the IoT-Agent) implements the following NGSI 9 and NGSI 10 operations.

A full description of NGSI9 and NGSI10 operations can be found here.

4.3.3.1 IoT-Agent incoming NGSI Messages
• queryContext (NGSI 10): The IoT-Agent receives a queryContextRequest. It generates in turn a query of last notice to IDAS database.
- **notifyContext (NGSI 10):** The IoT-Agent receives a notifyContextRequest. Embedded information is parsed and generate a command to IDAS

### 4.3.3.2 IoT-Agent outgoing NGSI Messages

- **registerContext (NGSI 9):** The IoT-Agent sends a registerContextRequest with the information received from IDAS once it has received in turn a registration operation from a Sensor (SensorML/SML RegisterSensor).
- **updateContext (NGSI 10):** The IoT-Agent sends an updateContextRequest with the information received from IDAS whenever a sensor has sent an observation (SML InsertObservation).

### 4.3.4 ETSI-M2M API

This API is exposed by the ETSI-M2M-Agent connector component. This connector enables ETSI-M2M communication between Gateway Device Management and Backend Device Management.

#### 4.3.4.1 Registering an Application

Request to Backend Dev Man:

```
POST /m2m/applications
Content-Type: application/json
{
    "application": {
        "appId": "myApp"
    }
}
```

Response:
4.3.4.2 Creating a Container

Request to Backend Dev Man:

POST /m2m/applications/myApp/containers

Content-Type: application/json

{  
    "container": {  
        "id": "temperature"  
    }  
}

Response:

201 Created
Location: /m2m/applications/myApp/containers/temperature

4.3.4.3 Pushing a Measure

Request to Backend Dev Man:
POST
/m2m/applications/myApp/containers/temperature/contentInstances

Content-Type: application/json
{
"degrees": 20
}

Response:

201 Created
Location:
/m2m/applications/myApp/containers/temperature/contentInstances/contentInstance2705559
5 Gateway Device Manager - OpenMTC - User and Programmers Guide

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

5.1 Introduction

This document describes the necessary steps to develop a software application Gateway Device Management functionality.

The Gateway Device Management GE is contains much of the "core" gateway functionality. It 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 in the ETSI M2M format (marshal/unmarshal). 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 is 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 Gateway Device Management GE is provided through the OpenMTC software developed by Fraunhofer FOKUS. The OpenMTC platform has been designed to act as a horizontal convergence layer or Machine-to-Machine (M2M) middleware for machine type communication that supports multiple vertical application domains. Those domains are usually the classic M2M verticals (market segments) such as transport and logistics, utilities, automotive, eHealth, etc which can be deployed independently or as part of a common platform.

5.1.1.1 Background and Detail

This User and Programmers Guide relates to the Gateway Device Management GE which is part of the Internet of Things (IoT) Services Enablement Architecture. Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.IoT.Gateway.DeviceManagement.
5.2 User’s Guide

Not applicable, since the Gateway Device Management GE's functionality is only accessible through programmatic interfaces (REST APIs).

5.3 Programmer's Guide

5.3.1 Interfaces
The Gateway Device Management GE implements the NGSI9 and NGSI10 interfaces according to the FI-WARE NGSI-9 and NGSI-10 RESTful API Specification as described here:


These interfaces are exposed at these URIs:

- [http://{serverRoot}:5050/NGSI9](http://{serverRoot}:5050/NGSI9)
- [http://{serverRoot}:5050/NGSI10](http://{serverRoot}:5050/NGSI10)

In addition, the raw restful ETSI M2M API (mId reference point) can be used. The API is described here:

**ETSI M2M mId Open RESTful API Specification (PRELIMINARY)**

The sclBase resource of the GSCLs resource tree is exposed here:

- [http://{serverRoot}:5000/m2m](http://{serverRoot}:5000/m2m)

5.3.2 Usage

5.3.2.1 **NGSI-9/10**
Being a RESTful interface, GE's interface can be used with any HTTP/1.1 compatible client software.
This is an example for calling the registerContext operation defined as part of NGSI-9:

The example uses the curl tool to issue a POST request to the URL which corresponds to the registerContext operation (/ngsi9/registerContext) with the XML representation of a registerContextRequest object as the request's body.

```
$ curl {serverRoot}/ngsi9/registerContext --header 'Content-Type: application/xml' -d '  
  <?xml version="1.0" encoding="UTF-8"?>
  <registerContextRequest>
    <contextRegistrationList>
      <contextRegistration>
        <entityIdList>
          <entityId type="" isPattern="false/true">  
            <id></id>
          </entityId>
        </entityIdList>
      </contextRegistration>
    </contextRegistrationList>
    <contextRegistrationAttributeList>
      <contextRegistrationAttribute>
        <name></name>
        <type></type>
        <isDomain></isDomain>
        <metaData>
          <contextMetadata>
            <name></name>
            <type></type>
            <value></value>
          </contextMetadata>
        </metaData>
      </contextRegistrationAttribute>
    </contextRegistrationAttributeList>
  </registerContextRequest>
```
The response to this operation is registerContextResponse object. Its XML representation looks as follows:

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

Very important here to save the REGISTRATION_ID together with the registered entities 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.
5.3.2.2 **ETSI M2M mld interface**

The Gateway Device Management GE implements the ETSI M2M interfaces according to the FI-WARE ETSI M2M mld Open RESTful API Specification as described here:

[ETSI M2M mld Open RESTful API Specification (PRELIMINARY)](https://example.com)

The usage of the mld reference point is further described in ETSI TS 102.690 and ETSI TS 102.921 respectively. Note that only the JSON representation of resources as specified by ETSI is supported.
6 Gateway Protocol Adapter - ZPA - User and Programmers Guide

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

6.1 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 standalone the client could be be realized either using the NGSI Northbound Interface or using the Generic Device Access Interface. The instructions contained in the “Developer Guide” section describe both these clients.

6.1.1.1 Background and Detail

This User and Programmers Guide relates to the Backend Device Management GE, which is part of the Internet of Things (IoT) Services Enablement Architecture. Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.IoT.Gateway.ProtocolAdapter.
6.2 User’s Guide

6.2.1 NGSI Support

The NGSI Support is essentially a module in the ZPA that allows the registration and the forwarding of the attribute values of NGSI-9 Context Entities (which model the IoT ZigBee devices) previously registered. In this context the ZPA represents an NGSI Event Producer and the interaction with this module can be done via a REST web service: in particular a servlet (representing an NGSI Event Consumer), which receives the incoming NGSI requests (registerContext and updateContext), is available at the URL "http://130.206.81.56:8080/zpa". The operation resources enabled are the following:

**Operation Resources for NGSI Northbound Interface**

<table>
<thead>
<tr>
<th>Resource Path</th>
<th>Description</th>
<th>Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/NGSI9/registerContext</td>
<td>A new device is discovered in the network</td>
<td>• POST</td>
<td>• registerContext</td>
</tr>
<tr>
<td>/NGSI10/updateContext</td>
<td>Event from a device already announced</td>
<td>• POST</td>
<td>• updateContext</td>
</tr>
</tbody>
</table>

Possible use cases and relations between them and the NGSI operations can be established:

<table>
<thead>
<tr>
<th>Use Case</th>
<th>ZPA behaviour</th>
<th>NGSI operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>receipt of the temperature by a sensor</td>
<td>the ZPA sends the value of the temperature whenever it is notified by the device</td>
<td>updateContext (the value of the sensor is specified at the level of payload)</td>
</tr>
</tbody>
</table>

**Operation Resources for Admin Interface specific for Esper4FastData GEi**

The following operations are used in order to initialize the communication when the ZPA GEi interacts with the Esper4FastData GEi. They must be performed before the first registerContext. In particular the Gateway Protocol Adapter (ZPA) allows the DB restart and the CEP instantiation into the Gateway Data Handling (Esper4FastData_Servlet). So for each operation the prefix path to use for the resources is the following: http://<host_Esper4FastData_GE>/<name_version_service_Esper4FastData> (see the Catalogue for the Data Handling GE at this link: http://catalogue.fi-ware.eu/enablers/gateway-data-handling-ge-esper4fastdata at the tab 'Instances').
<table>
<thead>
<tr>
<th>Resource Path</th>
<th>Description</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>/admin/esper4fastdata</td>
<td>reset the GE</td>
<td>• DELETE</td>
</tr>
<tr>
<td>/cep/instance</td>
<td>start the CEP component within the Esper4FastData GEi</td>
<td>• POST</td>
</tr>
</tbody>
</table>

6.2.2 GDA Support

The GDA Support is essentially a module in the ZPA that manages the communication with the IoT ZigBee devices and allows:

- to handle a new IoT ZigBee device discovered in the network
- to read the parameter value
- to actuate the commands of the IoT ZigBee device

but it is not needed to have a GUI to start it because it is a standalone process. It is possible to emulate a client of the ZPA using the servlet at this URL "http://protoadapter.lab.fi-ware.eu:8080/zpa" that shows the IoT ZigBee devices announced, the related properties (state and power) and the associated actuation commands.

![ZigBee Protocol Adapter is running!](image)

6.3 Programmers Guide

6.3.1 Client using the NGSI Northbound Interface

In the context NGSI the Gateway Protocol Adapter represents an Event Producer with the following behaviour:

<table>
<thead>
<tr>
<th>Event managed by the ZPA</th>
<th>NGSI operation implemented by the ZPA</th>
</tr>
</thead>
</table>

D.5.4.3: FI-WARE Users and Programmers Guide 66
Announcement of a new device in the network | RegisterContext (POST HTTP)
---|---
Event from a device already announced | UpdateContext (POST HTTP)

The description of the 2 NGSI operations is the following:

- **registerContext (NGSI 9)** – registration of an entity which model the ZigBee device indentifying it by its MAC Address and specifying the name and the type of its attributes

**URL:**

```html
http://<host>/<ID Event Consumer>/NGSI9/registerContext
```

**Header:**

```plaintext
Content-Type: application/xml
```

**Body:**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<registerContextRequest>
  <contextRegistrationList>
    <contextRegistration>
      <entityIdList>
        <entityId type="SmartPlug" isPattern="false">
          <id>000D6F000277BCC0</id>
        </entityId>
      </entityIdList>
    </contextRegistration>
  </contextRegistrationList>
</registerContextRequest>
```
<contextRegistrationAttribute>
    <name>manufacturer</name>
    <type>xs:string</type>
    <isDomain>false</isDomain>
</contextRegistrationAttribute>

<contextRegistrationAttribute>
    <name>model</name>
    <type>xs:string</type>
    <isDomain>false</isDomain>
</contextRegistrationAttribute>

<contextRegistrationAttribute>
    <name>protocol</name>
    <type>xs:string</type>
    <isDomain>false</isDomain>
</contextRegistrationAttribute>

<contextRegistrationAttribute>
    <name>currentPower</name>
    <type>xs:float</type>
    <isDomain>false</isDomain>
</contextRegistrationAttribute>

<contextRegistrationAttribute>
    <name>currentState</name>
    <type>xs:integer</type>
    <isDomain>false</isDomain>
</contextRegistrationAttribute>

</contextRegistrationAttributeList>

<registrationMetaData>
• updateContext (NGSI 10)—forwarding of the attribute values of an entity previously registered

URL:

http://<host>/<ID Event Consumer>/NGSI10/updateContext

Header:

Content-Type: application/xml

Body:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<updateContextRequest>
  <contextElementList>
    <contextElement>
      <entityId type="SmartPlug" isPattern="false">
        <id>000D6F000277BCC0</id>
      </entityId>
    </contextElement>
  </contextElementList>
</updateContextRequest>
<attributeDomainName>SmartPlug</attributeDomainName>
  <contextAttributeList>
    <contextAttribute>
      <name>name</name>
      <type>xs:string</type>
    </contextAttribute>
    <contextValue>FlegPlug000D6F000277BCC0</contextValue>
  </contextAttribute>
  <contextAttribute>
    <name>manufacturer</name>
    <type>xs:string</type>
    <contextValue>it.telecomitalia.zigbee.pa.ZigBeeGenericDeviceImpl@13803ba</contextValue>
  </contextAttribute>
  <contextAttribute>
    <name>model</name>
    <type>xs:string</type>
    <contextValue>FlexPlug</contextValue>
  </contextAttribute>
  <contextAttribute>
    <name>protocol</name>
    <type>xs:string</type>
    <contextValue>ZigBee</contextValue>
  </contextAttribute>
  <contextAttribute>
    <name>currentPower</name>
    <type>xs:float</type>
  </contextAttribute>
A developer may implement an NGSI Event Consumer which can handle these NGSI messages. Currently the ZPA implements only the NGSI Event Producer role, in future releases it will also implement the role of NGSI Event Consumer in which case it will be able to respond to requests from the client, such as: actuation requests, configurations requests, notifications requests, etc.

6.3.2 Client using the Generic Device Access Interface

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: ClientZigBeePA class
- Click on the “Finish”
- Copy the following into OSGI-INF/client.xml in your project:

```xml
<?xml version="1.0"?>
<scr:component
xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
configuration-policy="optional" name="Tester">
  <implementation
class="it.telecomitalia.client.zigbee.pa.ClientZigBeePA"/>
  <reference bind="setNetworkManager" cardinality="1..1"
interface="it.telecomitalia.ah.hac.lib.ext.INetworkManager"
name="INetworkManager" policy="dynamic"
unbind="unsetNetworkManager"/>
  <reference bind="setHttpService" cardinality="1..1"
interface="org.osgi.service.http.HttpService"
name="HttpService" policy="dynamic"
unbind="unsetHttpService"/>
  <reference bind="setGenericDevice" cardinality="0..n"
interface="com.ericsson.deviceaccess.api.GenericDevice"
name="GenericDevice" policy="dynamic"
unbind="unsetGenericDevice"/>
</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 the GenericDevice, NetworkManager and HttpService services are registered with the Service Registry, DS will obtain a reference to the new service objects and supply them to our component using the specified methods. Likewise the unbind attribute is the name of a method that will be called by DS when the services we were using become unavailable. The interface attribute specifies the name of the interfaces 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 instances of 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:

```java
com.ericsson.deviceaccess.api,
org.apache.commons.logging
it.telecomitalia.ah.hac.lib.ext,
javax.servlet,
javax.servlet.http,
javax.servlet.resources,
or.org.apache.commons.logging,
or.osgi.framework,
or.osgi.service.cm,
or.osgi.service.http
```
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/fiware/index.php/Gateway_Protocol_Adapter_-_Installation_and_Administration_Guide#Standalone_installation](https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/Gateway_Protocol_Adapter_-_Installation_and_Administration_Guide#Standalone_installation). This code is commented in the following section.

```java
package it.telecomitalia.client.zigbee.pa;

/** Copyright (C) 2012 Telecom Italia S.p.A. **/
import it.telecomitalia.ah.hac.lib.ext.INetworkManager;
import java.io.IOException;
import java.io.Writer;
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;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Vector;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.osgi.service.http.HttpService;
import org.osgi.service.http.NamespaceException;
public class ClientZigBeePA extends HttpServlet {
    private static final long serialVersionUID = 1L;
```
private final static Log log = LogFactory.getLog(ClientZigBeePA.class);

private String servletUri = "/zpa";

private HashMap<String, GenericDevice> devices = new HashMap();

private static String initialUri = null;

INetworkManager netMgr = null;

public void setHttpService(HttpService httpService) {
    log.debug("setHttpService");
    try {
        httpService.registerServlet(servletUri, this, null, null);
    } catch (ServletException e) {
        log.error("setHttpService", e);
    } catch (NamespaceException e) {
        log.error("setHttpService", e);
    }
}

public void unsetHttpService(HttpService httpService) {
    log.debug("unsetHttpService");
    httpService.unregister(servletUri);
}

protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    GenericDevice dev = null;
    String prop1 = null, prop2 = null, cmd = null, label =null;
    Vector<String> header = new Vector<String>();
    String devId = null;
int currentTarget = -1, vprop = -1;
float currentPower = -1;
GenericDeviceService service;
Writer writer = resp.getWriter();
GenericDeviceAction action;
if (req.getRequestURI().endsWith("openNwk")) {
    try {
        netMgr.openNetwork(60);
    } catch (Exception e) {
        e.printStackTrace();
        writer.write("<html><body>Some problem occurred while opening the ZigBee network</body></html>" );
    }
    resp.setContentType("text/html");
    writer.write("<html><body>The ZigBee network will be open for 60 seconds</body></html>" );
    return;
}
if (req.getRequestURI().contains("SetTarget")) {
    System.out.println ("doGet parameter: SetTarget found");
    if ((devId = req.getParameter("id")) != null && (vprop = Integer.parseInt(req.getParameter("prop"))) != -1) {
        if(devices.containsKey(devId)) {
            dev = this.devices.get(devId);
            service = dev.getService("SwitchPower");
            service.setPower(vprop);
        }
    }
}
```java
action = service.getAction("SetTarget");
GenericDeviceProperties args = action.createArguments();
if (vprop == 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
            GenericDeviceProperties ActionResult = result.getValue();
        } else {
            // Action failed for some reason.
            System.err.println("Failed in action: " + result.getReason());
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```
```html
writer.write("<html><head></head><body>

writer.write("<script>")

writer.write("window.location.href="+""+initialUri+"""");
writer.write("</script></body></html>");

} else {

resp.setContentType("text/html");
writer.write("<html><head><script
type='text/javascript'>")

writer.write("function myCommandFunction(id,uri,
devId, cmd, propValue) {
alert(uri+'/'+cmd+'?id='+devId+'&prop='+propValue);

window.location.href=uri+'/'+cmd+'?id='+devId+'&prop='+propValue
;
}

writer.write("function checkAppliance(id) {var
elm0,elm1,elm2,elm3;var VISIBILITY_HIDDEN = 'none';" numRows=6
```

```java
writer.write("document.getElementById(elm2).style.display = SET_VISIBILITY;
document.getElementById(elm3).style.display = SET_VISIBILITY;\n");
writer.write("</script></head>\n
writeTop(writer);

int i = 0, j=0;
Iterator it = devices.values().iterator();
while(it.hasNext())
{
    dev = (GenericDevice) it.next();
    if ((service = dev.getService("SwitchPower")) != null) {
        GenericDeviceProperties properties = service.getProperties();
        prop1 = "CurrentTarget";
        currentTarget = properties.getIntValue("CurrentTarget");
        cmd = "SetTarget";
        label = "Execute SetTarget";
    }
    if ((service = dev.getService("PowerSensor")) != null) {
        GenericDeviceProperties properties = service.getProperties();
        prop2 ="CurrentPower";
        currentPower = properties.getFloatValue("CurrentPower");
    }

    initialUri = "http://"+req.getLocalAddr()+":"+req.getLocalPort()+req.getRequestURI();
```
header.addElement(wrapRow(
    wrapCell("SmartPlug "+(i+1))
    + wrapCheckBox(i)
    + wrapCellStyle(i,j,"Smart Plug "+dev.getURN())
    + wrapCellStyle(i,++j,prop1+": "+currentState)
    + wrapCellStyle(i,++j,prop2+": "+currentPower)
    + wrapCommand(i,++j,initialUri, dev.getURN(), cmd, label, currentState));
    j =0;
    i++;
})
String table = wrapTable(header);
writer.write(table);
writeBottom(writer);"}
private String wrapTable(Vector<String> v) throws IOException {
    String content = "";
    for (int i = 0; i < v.size(); i++)
        content = content + v.get(i);
    System.out.println("wrapTable "+content);
    return content;
}

private String wrapCell(String value) throws IOException {
    return "<td width='100'>" + value + "</td>";
}

private String wrapRow(String value) throws IOException {
    return "<tr>" + value + "</tr>";
}

private String wrapCheckBox(int i) throws IOException {
    return "<td width='30'>" +
            "<input type='checkbox' id='ck\"+i\"'
            name='ck_appliance' value='s0' onclick='javascript:
            checkAppliance("+i");'/>" +"</td>";
}

private String wrapCellStyle(int i, int j, String value) throws IOException {
    return "<td>" + "+" +"</td>";
}

private String wrapCommand(int i, int j, String uri, String devId, String cmd, String label, int propValue) throws IOException {

When an OSGI HttpService is available our servlet can be registered and uploaded in a browser with the following URL:

```
http://<host>:<port>/zpa/
```

where the host is the server where the servlet runs and the port is 8080 as default.

When a NetworkManager object is created and registered as an OSGi service, so the client of the Protocol Adapter can be notified and it is possible to open the ZigBee network to allow the join of ZigBee smart plugs to the ZigBee network (see the instructions to set up the smart plugs described in the installation sections at https://forge.fi-
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. At this point in the servlet (at the following URL: http://<host>:<port>/zpa/) the discovered IoT devices (alias "Smart Plug") are listed. If the user check one of these, its properties and the related action are shown.

In the following paragraph is explained how these properties are retrieved and how the actuation action is executed.

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>
</service>
```
<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>
  </parameter>
</properties>
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”):

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

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

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

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

```java
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:

```java
float value = properties.getFloatValue("CurrentPower");
```
7 Gateway Protocol Adapter - EPCGE - User and Programmers Guide

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

7.1 Introduction

7.1.1 Background and detail

This User and Programmers Guide relates to the Protocol Adapter GE which is part of the Internet_of_Things_(IoT)_Services_Enablement_Architecture.

Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.IoT.Gateway.ProtocolAdapter. You can find hereunder a first description to better understand how to use this Generic Enabler.

7.1.1.2 EPCGE overview

The EPC GE is the part of the IoT Protocol Adapters and is responsible for RFID Network interaction.

As a consumer EPC GE can be seen as a Web Service enabling traceability of mobile items based on RFID EPC unique identification.

Electronic Product code (EPC) from GS1, is a standard solution like barcode used in the industry to:

- uniquely identify items using a standard naming scheme
- retrieve data about products with standards protocols (namely DNS).
- handle, trace or authenticate products in the industry supply chain, dataware house, retail stores etc ...
- handle events from supply chain stakeholders, flowing through EPC Network.

7.1.1.3 Value-added of a EPC GE system

This protocol Adapter allows you to retrieve all data of the digital life of any RFID tag. Especially it operates across different organizations. It helps improve business process flexibility and reliability. It filters the events, merges data from many events and has to deal with event privacy and subscribing.
7.1.1.4 **The EPCGE library**

EPCGE relies upon the open-source Fosstrak library. There is a very active community of developers behind it.

Here are some technical facts about it:

- Is available under GPL v2
- Is available as a Java library (jar)
- Implement several EPC Global standards like EPCIS or ALE

7.1.1.5 **EPCGE Concepts**

Suppose you want to buy the brand new Toyota 4x4. With the car model EPC product reference, you can ask the EPC resolver detailed information on my future car (like user manual, new features etc...). Thanks to the EPC resolver and the EPC Object Naming Service (ONS) that allows to uniquely identify this product and the manufacturer web EPCIS server.

Suppose you have ordered via an Internet e-shop your brand new Toyota 4x4. The e-shop provides better prices but has no stock. The counter part is that you have to wait until your car arrives from Japan factory.

As soon as your car is ready for shipping from Japan, you receive an SMS with the EPC code number corresponding to the RFID tag of your car. From now on you can trace your car all the way long to your country and to your home. Thanks again to the EPC resolver allowing to ask Discovery servers and EPCIS servers around the world to get all the historical data of your car.

7.1.1.6 **Use case**

When looking at the simulated use case below you can see that for example a particular item (4514300.182770) will be sent from CLIENT1 in GENES to ISTANBUL and that this information will be available in EPCIS1 database. Then the item will be sent from ISTANBUL to AMSTERDAM, with information stored in EPCIS2.

Following the same procedure you can easily get the traceability data for the other code samples.
When launching the test application, with the following URL: [EPCGE](#) some EPC codes samples appear on the browser page. For example the first one is: urn:epc:id:sgtin:4514280.191438.1375091273041 This EPC code is an RFID tag ID corresponding to an EPC Serial Global trade Item Number (epc:id:sgtin), where 4514284 is the manufacturer reference, 198196 is the product reference and 1370337191191 the item serial number.

### 7.1.1.6.1 Functional and logical architecture

The EPC Rest Adaptor and EPC Resolver will be installed on the Fiware test bed. EPC services like Discovery service, EPCIS databases and ONS name servers are external servers accessed transparently through the resolver via UDP or SOAP.

The core part of EPC enabler is the resolver. This module will handle complex requests, enabling for example a mobile phone, after scanning of the 2D barcode on a fridge in my kitchen, to get a complete data history for that fridge. That is to say when and where it has been built, all the shipping and logistic information, until it finally arrived in my kitchen.
The complete process consists to request the ONS for that EPC code in order to get the so-called referent discovery server for that fridge. Then the resolver asks this referent discovery server to get the complete list of EPCIS servers and other discovery servers having data on that fridge.

Finally, the resolver will request all the servers of that list to rebuild the complete history of my fridge sorted by time of appearance and send back to the mobile phone the whole story.
7.1.1.6.2 Physical architecture

7.2 User’s Guide

Traceability of an EPC from a browser:

Clicking on the URL EPCGE will display the following home page:
The home page consists of four parts:

- The top part is a link to the WADL and a link to test the service with a predefined EPC code.
- The left part is an explorer to browse the different services
- The right part consist, on top to the query part and on the bottom to the response part
### 7.2.1.1 WADL File

Click on the WADL link to get XML WADL file:

```xml
- <doc jersey:generatedBy="Jersey 1.9.1 09/14/2011 02:05 PMG">
  - <include href="/application.wadl"/>
  - <doc title="Generated with Jersey"/>
</doc>
</resource>

- <resource path="/traces/tracingEPC">
  - <method id="traceDefaultEPC" name="GET">
    - <response>
      <representation mediaType="text/plain"/>
    </response>
  </method>
  - <method id="/codeEPC" name="/">
    <param name="/codeEPC" style="template" type="text:string"/>
  </method>
</resource>

- <resource path="/etl/tracingEPCObjectEvent">
  - <method id="create" name="POST">
    - <request>
      <ns2:representation element="etl:EventObjectEvent" mediaType="application/xml"/>
      <ns2:representation element="etl:EventObjectEvent" mediaType="application/json"/>
    </request>
  </method>
  - <method id="/findAI" name="/">
    - <response>
      <ns2:representation element="etl:EventObjectEvent" mediaType="application/xml"/>
      <ns2:representation element="etl:EventObjectEvent" mediaType="application/json"/>
    </response>
  </method>
</resource>
```

Click on the [Test to trace a predefined CodeEPC](#) (just below WADL link) to get an XML trace of the Get request on the predefined EPC code.

### 7.2.1.2 Tracing an EPC Code

Each and every `<EPCISEvent>` tag is an EPCIS Event involving the EPC code referenced in the `<epcs>` tag: `urn:epc:id:sgtin:4514280.191438.1375091273041`
Back to the home page, on the left side, you see an “explorer like” window that you can expand by clicking on the + sign.

Two possibilities:

- If there is no Internet connectivity, the “entities.eventobjectevent” entry allows you to request a local EPCIS database located on the Fiware testbed.

- The other entry “entities.tracingEPC” will use the local resolver on the testbed to discover over Internet all of the traceability data for a particular EPC code item.

Open the “entities.tracingEPC” and click on the “{codeEPC}” entry.
At that point you have 2 choices, http GET or http POST.

http GET: Choose GET option in "choose method to test"
Enter the code EPC to trace in the text form:

You can copy and paste an EPC code from the given code samples:

```
urn:epc:id:sgtin:4514280.191438.1375091273041
urn:epc:id:sgtin:4514300.182770.1375091275209
urn:epc:id:sgtin:4514284.198196.1375091273992
urn:epc:id:sgtin:4514274.109365.1375091275892
```

Choose “Get” request in the choose method to test and “application/xml” in the “MIME” field.
Then, click on:
All the related events to that EPC code, in XML format, will be displayed in the response part sorted in the chronological order showing, among other data, the event time and the readpoints.

By the way the “trace carto” button has become activated! So, to display results on the map, click on:
The result shows that this EPC item has been sent from Genes to Istanbul and then from Istanbul to Amsterdam (you can “hardly” see the arrow on Amsterdam on the picture above).

To remove all the traces from the map, click on the "clean traces" button.

If you choose “Get” request in the choose method to test and “application/json” in the “MIME” field and then, click on:

![Get Events](image)

All the related events to that EPC code, in JSON format, will be displayed in the response part sorted in the chronological order.

Nota: if you look in the text response part and compare the scroll bar between XML and JSON, you see that JSON is less verbose then XML.
Http POST

Choose POST option in "choose method to test"

Choose application/xml in "MIME"

Copy the following XML payload in the "Content" area:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
  <entityIdList>
    <entityId type = "test" isPattern="false">
      <id>urn:epc:id:sgtin:4514280.191438.1375091273041</id>
    </entityId>
  </entityIdList>
</queryContextRequest>
```
Click on the "Get Events" button, after a few seconds you should see the complete trace response in xml NGSI format:
7.3 Programmer’s Guide

The EPCGE component is a part of the Data Handling Generic Enabler. It features REST oriented interfaces. It also features a partial implementation of the OMA-NGSI 9&10 standard to communicate with the others components/GE. The following sections explains how to use the EPCGE Servlet as a programmer. The Gateway EPCGE Servlet exposes methods to get historical information from EPC objects through a Restful API via HTTP.

Traceability of an EPC from an application

It is possible to request the REST interface of the EPC enabler by executing a curl command or an http Get request. The command specifies the return format between JSON or XML.


Will return traceability data in JSON format: The answer, sorted by event time appearance, will look like this:
With HTTP GET

Requesting historical information for the item with EPC code urn:epc:id:sgtin:4514274.109365.1375091275892:


Will return traceability data in XML format: The answer, sorted by event time appearance, will look like this:

Result

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ePCISEvents>
  <EPCISEvent>
    <action>ADD</action>
    <bizLoc>urn:unicaen:iotatester:xxxx:ssl:_not_specified_</bizLoc>
    <bizTrans>null</bizTrans>
    <EPCClass>""</EPCClass>
    <epcs>urn:epc:id:sgtin:4514280.191438.1375091273041</epcs>
    <eventTime>"2012-08-06T14:43:02.280+02:00"</eventTime>
    <insertedTime>"2012-08-06T14:43:02.280+02:00"</insertedTime>
    <parentID>""</parentID>
    <quantity>""</quantity>
    <readPoint>urn:unicaen:iotatester:xxxx:ssl:3</readPoint>
    <type>"Object"</type>
  </EPCISEvent>
  ...
</ePCISEvents>
```
<disposition>urn:unicaen:iotatester:disp:xxxx:tester</disposition>

  <EPCClass/>

<epcs>urn:epc:id:sgtin:4514274.109365.1375091275892</epcs>

  <eventTime>2013-07-29T11:47:55.893+02:00</eventTime>

  <insertedTime>2013-07-29T11:47:55.893+02:00</insertedTime>

  <parentID/>

  <quantity/>

<readPoint>urn:unicaen:iotatester:xxxx:ssl:3</readPoint>

  <type>Object</type>

  </EPCISEvent>

  <EPCISEvent>

    <action>ADD</action>

    <bizLoc>urn:unicaen:iotatester:xxxx:ssl:_not_specified_</bizLoc>


    <bizTrans/>

    <disposition>urn:unicaen:iotatester:disp:xxxx:tester</disposition>

    <EPCClass/>

<epcs>urn:epc:id:sgtin:4514274.109365.1375091275892</epcs>

  <eventTime>2013-07-29T11:48:03.095+02:00</eventTime>

  <insertedTime>2013-07-29T11:48:03.095+02:00</insertedTime>

  <parentID/>
With HTTP POST

Requesting historical information for the item with EPC code urn:epc:id:sgtin:4514274.109365.1375091275892:
post request:
curl 161.105.138.141:8080/EPCGE-TESTBED-NGSI/webresources/entities/querycontext -s -S --header 'Content-Type: application/xml' -d @- <<EOF
XML payload:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
    <entityIdList>
        <entityId type = "test" isPattern="false">
            <id>urn:epc:id:sgtin:4514280.191438.1375091273041</id>
        </entityId>
    </entityIdList>
    <attributeList>
        <attribute>action</attribute>
    </attributeList>
</queryContextRequest>
EOF

post response:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<queryContextResponse>
    <contextResponseList>
        <contextElementResponse>
            <contextElement>
                <entityId isPattern="false">
                    <id>urn:epc:id:sgtin:4514280.191438.1375091273041</id>
                </entityId>
            </contextElement>
        </contextElementResponse>
    </contextResponseList>
</queryContextResponse>
```
<bizLoc>urn:unicaen:iotatest:xxxx:ssl:_not_specified_</bizLoc>


    <bizTrans/>

<disposition>urn:unicaen:iotatest:disp:xxxx:tester</disposition>

    <EPCClass/>

<epcs>urn:epc:id:sgtin:4514280.191438.1375091273041</epcs>

    <eventTime>2013-07-29T11:47:54.204+02:00</eventTime>

    <insertedTime>2013-07-29T11:47:54.204+02:00</insertedTime>

    <parentID/>

    <quantity/>

<readPoint>urn:unicaen:iotatest:xxxx:ssl:10</readPoint>

    <type>Object</type>

    </contextValue>

</contextAttribute>

<contextAttribute>

    <name>Event</name>

    <contextValue>

        <action>ADD</action>

</contextValue>

</contextAttribute>

<bizLoc>urn:unicaen:iotatest:xxxx:ssl:_not_specified_</bizLoc>


    <bizTrans/>
<disposition>urn:unicaen:iotatest:disp:xxxx:tester</disposition>

    <EPCClass/>

<epcs>urn:epc:id:sgtin:4514280.191438.1375091273041</epcs>

    <eventTime>2013-07-29T11:47:54.405+02:00</eventTime>

    <insertedTime>2013-07-29T11:47:54.405+02:00</insertedTime>

    <parentID/>

    <quantity/>

<readPoint>urn:unicaen:iotatest:xxxx:ssl:5</readPoint>

    <type>Object</type>

    </contextValue>

    </contextAttribute>

    </contextAttributeList>

    </contextElement>

    </contextElementResponse>

    </contextResponseList>

    </queryContextResponse>

7.3.1 Detailed EPC Reference

See [GS1 Home page EPCglobal Standards Overview](#)

7.3.2 Glossary of terms

- **Event**: anything that happens

- **Event Object, event message**: an object that represents 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.

- Electronic Product Code (EPC): GS1 standard that uniquely identify items using a standardized naming scheme.
8 Gateway Protocol Adapter – MRCoAP – User and Programmers Guide

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

8.1 Introduction

The MRCoAP Adapter is an implementation of the Protocol Adapter GE and is the glue between NGSI and a moterunner 6LoWPAN/CoAP WSN (Wireless Sensor Network) that is composed by the target device(s). Through the MRCoAP Adapter a client is able to discover devices, get parameter readings and execute commands to actuate them.

The expected behavior of the MRCoAP is based on these phases:

- Device discovery
- Device parameter read

It is possible to get the value of a parameter of a device

- Device actuation

8.2 User’s Guide

8.2.1 NGSI Support

The NGSI allows the registration and the forwarding of the attribute values of NGSI-9 Context Entities (which model the IoT ZigBee devices) previously registered. In this context the MRCoAP represents an NGSI Event Producer and the interaction with this module can be done via a REST web service: in particular a servlet (representing an NGSI Event Consumer), which receives the incoming NGSI requests (registerContext and updateContext)

**Operation Resources for NGSI Northbound Interface**

<table>
<thead>
<tr>
<th>Resource Path</th>
<th>Description</th>
<th>Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/NGSI9/registerContext</td>
<td>A new device is discovered in the network</td>
<td>• POST</td>
<td>• registerContext</td>
</tr>
<tr>
<td>/NGSI10/updateContext</td>
<td>Event from a device already announced</td>
<td>• POST</td>
<td>• updateContext</td>
</tr>
</tbody>
</table>

Possible use cases and relations between them and the NGSI operations can be established:
## 8.3 Programmers Guide

### 8.3.1 Client using the NGSI Northbound Interface

In the context NGSI the Gateway Protocol Adapter represents an Event Producer with the following behaviour:

<table>
<thead>
<tr>
<th>Event managed by the MRCoAP</th>
<th>NGSI operation implemented by MRCoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announcement of a new device in the network</td>
<td>RegisterContext (POST HTTP)</td>
</tr>
<tr>
<td>Event from a device already announced</td>
<td>UpdateContext (POST HTTP)</td>
</tr>
</tbody>
</table>

The description of the 2 NGSI operations is the following:

- **registerContext (NGSI 9)** – registration of an entity which model the ZigBee device indentifying it by its MAC Address and specifying the name and the type of its attributes

- **updateContext (NGSI 10)** – forwarding of the attribute values of an entity previously registered

A developer may implement an NGSI Event Consumer which can handle these NGSI messages. Currently MRCoAP implements only the NGSI Event Producer role.
9 Gateway Data Handling - Esper4FastData - User and Programmers Guide

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

9.1.1 Introduction

This User and Programmers Guide is intended for EspR4FastData 3.3.3 version.

9.1.1.1 Background and detail

This User and Programmers Guide relates to the Gateway Data Handling GE which is part of the Internet_of_Things_(IoT)_Services_Enablement_Architecture. Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.IoT.Gateway.DataHandling. You will find hereunder first concepts to better understand how to use this Generic Enabler.

9.1.1.2 Value-added of the EspR4FastData Application

Overall, the main role of the EspR4FastData GEi is to perform the following, in real-time:

- Data filtering
- Data aggregation
- Data merging

In order to achieve this, three main features are available:

- Event types management: setting input event data structures.
- Rules management: configuring the way event data are processed, including calculations, conditions, thresholds...
- Event sinks management: configuring where to propagate processed output events.

9.1.1.3 Detailed EPL Reference

The EspR4FastData application heavily relies on the open-source Esper (by EsperTech) library. EspR4FastData event processing rules are written using the EPL language, which is an SQL dialect.

9.1.1.4 **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

9.2 **User's Guide**

Data Handling GE runs as a servlet within a gateway. It does not feature a Graphical User Interface (GUI). It is accessed through its REST API, described in the Programmer's Guide section.

9.3 **Programmer's Guide**

9.3.1 **Introduction**

The EspR4FastData component is a version of the Data Handling Generic Enabler. It features a REST oriented API, among which a partial implementation of the OMA-ngsi 9&10 standard is provided.

This manual emphasizes a practical approach in order to facilitate understanding. **It is presented in the form of a tutorial: a concrete use case is proposed.**
The following sections explain how to use EspR4FastData as a programmer.

### 9.3.1.1 Steps To Follow Before Starting The Tutorial

Before anything else, please follow these installation instructions.

Then get the SoapUI software. Install it, open the "EspR4FastData-soapui-project-3.3.3.xml" SoapUI project, and expand the "QuickStart" test suite:

```
- EspR4FastData-3.3.3
  - EventSink-API
  - CEP management
  - NgsI10-NgsI10 features
  - Parcel delivery example
- Quickstart (see the Users and Programmers guide)
```

Double click on the "QuickStart" test case. It opens a related detailed window. Click on the "URL" button then choose "localhost" as target URL. This sets the root URL as "http://localhost".

This User and Programmers Guide assumes you are running all softwares on the same host, on the local network interface.

Double click on the "QuickStart" test case. It opens a related detailed window. Click on the "URL" button then choose "localhost" as target URL. This sets the root URL as "http://localhost".
You are now ready to begin.

9.3.1.2 **Text Color Conventions**
- Requests messages are in blue
- Response messages are in green

9.3.1.3 **Quickstart Tutorial Scenario: Parcel Delivery Use Case**
The goal of this example is to highlight the main EspR4FastData features by merging cross-business real-time events. This almost explores the whole EspR4FastData REST API. When an existing feature is not needed in this use-case scenario, it is detailed apart.

This scenario is about a geolocated parcel that is being delivered by different sequenced means of transport.

EspR4FastData allows to localize the parcel in real time, by configuring it with a simple merging CEP rule.

At any point of time, the parcel is carried by a mean of transport (a plane, a first van, then a second van). These vehicles send their GPS coordinates in real-time to EspR4FastData. The parcel state is traced as well.

This scenario involves two entity types: "ParcelMonitoring" and "VehicleLocation".

The vehicle entities ("Van1", "Van2" and "Plane" being the vehicle idetifiers) are meant to be localized in real-time, and as such, they belong to the "VehicleLocation" type, which features the following two properties: "latitude" and "longitude".

A "Parcel0" ("Parcel0" being the parcel identifier) entity is always linked to a particular vehicle entity at any given moment. The "Parcel0" entity belongs to the "ParcelMonitoring" type, which features the two following properties: "deliveryStep" (which could be like "ordered", "delivery in progress", "delivered"...etc), and "vehicleId" which is the identifier of the vehicle in which the parcel is being transported.
A simple EspR4FastData statement allows the merging of events from a given parcel, and the events from transportation vehicles.

The resulting merged entity is "LocalizedParcel0", which features both the properties from the "ParcelMonitoring" and "VehicleLocation" type.

![Diagram showing the merging of events from a parcel and transportation vehicles]

It should be noted that for practical reasons, only two events (one for the parcel, and one for the vehicles) are fired and merged. The vehicles are supposed to send their GPS coordinates at a rate that does not fit this manual.

When a paragraph or a sentence is about the tutorial, it is written in italic type. The tutorial is divided into 18 steps and it starts here.

9.3.2 Application Level Administration Tutorial

9.3.2.1 REST Resources Overview

- http://{ServerRoot}/EspR4FastData-3.3.3/admin/espr4fastdata
- http://{ServerRoot}/EspR4FastData-3.3.3/admin/proxy
- http://{ServerRoot}/EspR4FastData-3.3.3/admin/backupObjects/contextRegistrations

9.3.2.2 Resource admin/espr4fastdata

9.3.2.2.1 HTTP Delete Method

Tutorial / STEP 1: we need at first to completely reset EspR4FastData, to make sure there are no residual data left. For this, we use admin/espr4fastdata REST resource:

- **DESCRIPTION**: This operation completely destroy the CEP threads, the CEP data, and all the application level data. It can be used whether the application is running or not. It leaves the application clean and ready to start.
- **INPUT**: No payload / No URL parameters.
- **RETURN**: ResponseMessage data structure.

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/admin/espr4fastdata
(queried with HTTP Delete method)
```

**Example Payload**

```
No payload.
```

**Example XML Response**

```
<responseMessage>
  <code>200</code>
  <reasonPhrase>OK</reasonPhrase>
</responseMessage>
```

**Tutorial / STEP 2**: run the same operation on [http://localhost/EventSink](http://localhost/EventSink) in order to also reset the dummy event sink (recipient).

Then we will [register the "ParcelMonitoring" and "VehicleLocation" types](http://localhost/EventSink).

---

9.3.2.3 **Resource admin/proxy**

9.3.2.3.1 **Http Post Method**

- **DESCRIPTION**: In order to allow output events to pass a proxy, this operation allows proxy configuration.
- **INPUT**: ProxyParams XML data structure featuring the proxy URL and proxy TCP port.
- **RETURN**: ProxyParams XML data structure.

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/admin/proxy
(queried with HTTP Post method)
```

**Example Payload**

```
```
Example XML Response

```xml
<proxyParams>
  <uri>p-goodway.rd.francetelecom.fr</uri>
  <port>3128</port>
</proxyParams>
```

### 9.3.2.3.2 **Http Get Method**

- **DESCRIPTION:** This operation reads and returns the current proxy configuration.
- **INPUT:** No payload / No URL parameters
- **RETURN:** ProxyParams XML data structure.

**Example URL**

http://localhost/EspR4FastData-3.3.3/admin/proxy (queried with HTTP Get method)

**Example Payload**

No payload.

**Example XML Response**

```xml
<proxyParams>
  <uri>p-goodway.rd.francetelecom.fr</uri>
  <port>3128</port>
</proxyParams>
```
9.3.2.4  

**Resource admin/backupObjects/{objectName}**

9.3.2.4.1  

**Http Get Method**

- **DESCRIPTION**: Low level access to application state and application-wide persisted objects.
- **INPUT**: The requested object name is provided in the URL as a path parameter.
- **RETURN**: The raw object content.

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/admin/backupObjects/contextRegistrations (queried with HTTP GET method)
```

**Example XML Response**

```xml
<contextRegistrationList>
   <contextRegistration>
      <RegistrationId>465a7021-bf5c-4070-a282-5dd913eaf1b9</RegistrationId>
      <entityIdList>
         <entityId isPattern="false" type="ParcelMonitoring">
            <id>Parcel0</id>
         </entityId>
      </entityIdList>
   </contextRegistration>
   <contextRegistration>
      <name>vehicleId</name>
      <type>xs:string</type>
      <isDomain>false</isDomain>
   </contextRegistration>
   <contextRegistration>
      <name>deliveryStep</name>
      <type>xs:string</type>
   </contextRegistration>
</contextRegistrationList>
```
9.3.3 ngsi-9 Features Tutorial

ngsi is a standard that describes an abstract interface. It can be implemented to describe almost any "thing" and its related properties. It also describes publish-subscribe features. FiWare has implemented this interface in a REST manner, using XML as binding.

9.3.3.1 REST Ressources Overview

Only one resource is available for the time being.

- http://{ServerRoot}/EspR4FastData-3.3.3/ngsi9/registerContext

9.3.3.2 Resource ngsi9/registerContext

9.3.3.2.1 Http Post Method

Tutorial / STEP 3-4: After starting the CEP component, we are going to register the use-case entities and their related types.
• **DESCRIPTION**: This operation registers an ngsi context within EspR4FastData. This makes an input event type and related entities to be known from an EspR4FastData perspective. An event is characterized by the involved entities (e.g., Van1, Van2, Plane), its type, and properties that related to the type. For an event to be processed, it is mandatory to be "known" by the application., otherwise it would be rejected.

• **INPUT**: The RegisterContextRequest payload describes an event type, its related attributes and the entities belonging to this type. It also features a "duration" field, which limits the registration to a maximum duration. Each ContextRegistration data structure features a "providingApplication" field which links to the event provider (the application that pushes events towards EspR4FastData). It is possible to register many entities in the same request, but in such case they must belong to the same type. This case is illustrated in the XML payload sample below.

• **RETURN**: The RegisterContextResponse XML data structure features the duration and a registration Id, and also an XSD message in the "details" field. This XSD structure is a translation of the RegisterContextRequest XML message into something that is understandable by the internal CEP component. For event descriptions, the internal CEP component only understands XSD, so the ngsi registration message must be internally translated this way.

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/ngsi9/registerContext
(queried with HTTP Post method)
```

**Tutorial / STEP 3**: we are going to register the "VehicleLocation" type, along with the "Plane", "Van1", and "Van2" entities.

**registerContext "VehicleLocation" Type Payload**

```
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
  <contextRegistrationList>
    <contextRegistration>
      <entityIdList>
        <entityId type="VehicleLocation" isPattern="false">
          <id>Plane</id>
        </entityId>
      </entityIdList>
    </contextRegistration>
  </contextRegistrationList>
</registerContextRequest>
```
<entityId type="VehicleLocation" isPattern="false">
  <id>Van1</id>
</entityId>

<entityId type="VehicleLocation" isPattern="false">
  <id>Van2</id>
</entityId>

<contextRegistrationAttributeList>
  <contextRegistrationAttribute>
    <name>latitude</name>
    <type>xs:double</type>
    <isDomain>false</isDomain>
  </contextRegistrationAttribute>
  <contextRegistrationAttribute>
    <name>longitude</name>
    <type>xs:double</type>
    <isDomain>false</isDomain>
  </contextRegistrationAttribute>
</contextRegistrationAttributeList>

<registrationMetaData>
  <contextMetadata>
    <name>timestamp</name>
    <type>xs:dateTime</type>
    <value/>
  </contextMetadata>
</registrationMetaData>
"registerContext "VehicleLocation" Type XML Response"

```xml
<registerContextResponse>
  <duration>P5D</duration>

<registrationId>737E0B92DA0586329378E4B40C5B6FC4</registrationId>

<errorCode>
  <code>200</code>
  <reasonPhrase>OK</reasonPhrase>
  <details><![CDATA[Event type "VehicleLocation" has been created:]]>
</details>

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="VehicleLocation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="entityIdId" type="xs:string" />
        <xs:element name="attributeDomainName" type="xs:string" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Tutorial / STEP 4: we are going to register the "ParcelMonitoring" type, along with the "Parcel0" entity.

**registerContext "ParcelMonitoring" Type Payload**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<registerContextRequest>
    <contextRegistrationList>
        <contextRegistration>
            <entityIdList>
                <entityId type="ParcelMonitoring" isPattern="false">
                    <id>Parcel0</id>
                </entityId>
            </entityIdList>
            <contextRegistrationAttributeList>
                <contextRegistrationAttribute>
                    <name>vehicleId</name>
                    <type>xs:string</type>
                </contextRegistrationAttribute>
            </contextRegistrationAttributeList>
        </contextRegistration>
    </contextRegistrationList>
</registerContextRequest>
```
<isDomain>false</isDomain>
</contextRegistrationAttribute>
<contextRegistrationAttribute>
  <name>deliveryStep</name>
  <type>xs:string</type>
  <isDomain>false</isDomain>
</contextRegistrationAttribute>
</contextRegistrationAttributeList>
<registrationMetaData>
  <contextMetadata>
    <name>timestamp</name>
    <type>xs:dateTime</type>
    <value></value>
  </contextMetadata>
</registrationMetaData>
</registrationMetaData>

<providingApplication>http://localhost/EspR4FastData-3.3.3</providingApplication>
</contextRegistration>
</contextRegistrationList>
<duration>P5D</duration>
<registrationId/>
</registerContextRequest>

registerContext "ParcelMonitoring" Type XML Response

<registerContextResponse>
  <duration>P5D</duration>
</registerContextResponse>
<registrationId>C74B46F7C61DD9223D2C64B38EEAA3CD</registrationId>

<errorCode>
  <code>200</code>
  <reasonPhrase>OK</reasonPhrase>
  <details><![CDATA[Event type "ParcelMonitoring" has been created:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="ParcelMonitoring">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="entityIdId" type="xs:string"/>
        <xs:element name="attributeDomainName" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="vehicleId" type="xs:string"/>
        <xs:element name="deliveryStep" type="xs:string"/>
        <xs:element name="metadata_timestamp" type="xs:dateTime"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>]]></details>
</errorCode>
</registerContextResponse>
Tutorial STEP 3-4: we can see that an ngsi registration also implies the creation of an event type inside the CEP component. This event type must be described with XSD for the CEP to understand it. The translation from ngsi to XSD is internally automated.

To summarize, here is what we have done using the ngsi-9 registerContext operation:

- we have created two event types which are "VehicleLocation" and "ParcelMonitoring".
- we have created the properties/attributes associated with the new event types: properties "latitude" and "longitude" are associated to the "VehicleLocation" event type, whereas "vehicleId" and "deliveryStep" are associated to the "ParcelMonitoring" event type. The "timestamp" meta-data is related to the whole event type. It should be noted that "timestamp" is not a reserved keyword. It is just here because a time stamp is needed in the use-case.
- we have associated entities to event types: "Parcel0" belongs to the "ParcelMonitoring" type, whereas "Plane", "Van1" and "Van2" belong to the "VehicleLocation" event type.

Now we are going to read all available event types in the system, in order to confirm that new event types have been created according to the registered context properties.

9.3.4 ngsi-10 Features Tutorial

ngsi is a standard that describes an abstract interface. It can be implemented to describe almost any "thing" and its related properties. It also describes publish-subscribe features. FiWare has implemented this interface in a REST manner, using XML as binding.

9.3.4.1 REST Ressources Overview

- http://{ServerRoot}/EspR4FastData-3.3.3/ngsi10/updateContext
- http://{ServerRoot}/EspR4FastData-3.3.3/ngsi10/queryContext
- http://{ServerRoot}/EspR4FastData-3.3.3/ngsi10/subscribeContext
- http://{ServerRoot}/EspR4FastData-3.3.3/ngsi10/unsubscribeContext

9.3.4.2 Resource ngsi10/updateContext

9.3.4.2.1 Http Post Method

Tutorial / STEP 12-13: We now have completely configured EspR4FastData in order to make it ready to process incoming events: types and entities have been registered, and a rule
statement has been configured. At first, let's send an event that sets the "Parcel0" state to "DeliveryInProgress" and links it to the "Van1" vehicle.

- **DESCRIPTION:** Implementation of the OMA-ngsi updateContext operation. In EspR4FastData, this resource receives events coming from event producers. Please check the FiWare common ngsi model and the official OMA documentation as well.
- **INPUT:** UpdateContextRequest XML data structure.
- **RETURN:** UpdateContextResponse XML data structure.

Example URL

```
http://localhost/EspR4FastData-3.3.3/ngsi10/updateContext
(queried with HTTP Post method)
```

Tutorial / STEP 12: We send the "Parcel0" event which is related to the "ParcelMonitoring" event type.

"ParcelMonitoring" Event Example Request Payload

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<updateContextRequest>
  <contextElementList>
    <contextElement>
      <entityId type="ParcelMonitoring" isPattern="false">
        <id>Parcel0</id>
      </entityId>
    </contextAttribute>
    <contextAttribute>
      <name>vehicleId</name>
      <type>string</type>
      <contextValue>Van1</contextValue>
    </contextAttribute>
    <contextAttribute>
      <name>deliveryStep</name>
      <type>string</type>
    </contextAttribute>
  </contextElementList>
</updateContextRequest>
```
"ParcelMonitoring" Event Response Payload

<updateContextResponse>
  <errorCode>
    <code>200</code>
    <reasonPhrase>OK</reasonPhrase>
  </errorCode>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId isPattern="false" type="ParcelMonitoring">
          <id>Parcel0</id>
        </entityId>
      </contextElement>
    </contextElementResponse>
  </contextResponseList>
</updateContextResponse>
<contextAttributeList>
  <contextAttribute>
    <name>vehicleId</name>
    <type>string</type>
    <contextValue>Van1</contextValue>
  </contextAttribute>
  <contextAttribute>
    <name>deliveryStep</name>
    <type>string</type>
    <contextValue>DeliveryInProgress</contextValue>
  </contextAttribute>
</contextAttributeList>
<domainMetadata>
  <contextMetadata>
    <name>timestamp</name>
    <type>string</type>
    <value>20130301T140609+0100</value>
  </contextMetadata>
</domainMetadata>
</contextElement>
<statusCode>
  <code>200</code>
  <reasonPhrase>OK</reasonPhrase>
</statusCode>
</contextElementResponse>
</contextResponseList>
Tutorial / STEP 13: We send the "Van1" event which belongs to the "VehicleLocation" event type, with "latitude" and "longitude" properties.

"VehicleLocation" Event Example Request Payload

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<updateContextRequest>
    <contextElementList>
        <contextElement>
            <entityId type="VehicleLocation" isPattern="false">
                <id>Van1</id>
            </entityId>
            <contextAttributeList>
                <contextAttribute>
                    <name>latitude</name>
                    <type>double</type>
                    <contextValue>43.665512</contextValue>
                </contextAttribute>
                <contextAttribute>
                    <name>longitude</name>
                    <type>double</type>
                    <contextValue>7.20353</contextValue>
                </contextAttribute>
            </contextAttributeList>
            <domainMetadata>
                <contextMetadata>
                    <name>timestamp</name>
                    <type>string</type>
                </contextMetadata>
            </domainMetadata>
        </contextElement>
    </contextElementList>
</updateContextRequest>
```
"VehicleLocation" Event Response Payload

```xml
<updateContextResponse>
  <errorCode>
    <code>200</code>
    <reasonPhrase>OK</reasonPhrase>
  </errorCode>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId isPattern="false" type="VehicleLocation">
          <id>Van1</id>
        </entityId>
        <contextAttributeList>
          <contextAttribute>
            <name>latitude</name>
            <type>double</type>
            <contextValue>43.665512</contextValue>
          </contextAttribute>
        </contextAttributeList>
      </contextElement>
    </contextElementResponse>
  </contextResponseList>
</updateContextResponse>
```
Tutorial / STEP 12-13: It is important to understand what happens when this last "VehicleLocation" event is processed, because it is not so obvious, albeit not very complicated. These two "VehicleLocation" and "ParcelMonitoring" events have the same timestamp, and it appears that ParcelMonitoring.vehicleId = VehicleLocation.entityIdId (check again the related CEP rule statement to remember it). The rule statement triggering condition is thus fulfilled: an event featuring the merged "LocalizedParcel0" entity is fired internally. Of course it is necessary to register it to the event sink prior to sending any related event to this sink. Thus a "registerContext" payload is automatically generated and pushed towards it. Finally, an updateContext (event) payload featuring the "LocalizedParcel0" and properties is then propagated towards the sink.
As a STEP 14 activity, check again http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement/eventsinkurls to find that the <ngsiRegistered> element is now set to “true”. It means that registering to the event sink was done successfully.

We are almost done with this tutorial. The output event has now been processed and stored in EspR4FastData, as always when a rule statement is triggered. Let’s check the stored event by calling the queryContext resource.

9.3.4.3 Resource ngsi10/queryContext

9.3.4.3.1 Http Post Method
Tutorial / STEP 15: This is the last step. We are going to query the “LocalizedParcel0” entity to retrieve the previous event data.

- **DESCRIPTION**: Implementation of the OMA-ngsi queryContext operation. This operation returns an entity that is related to a processed event. After processing, each event is stored in order to keep its related ngsi context up-to-date. Please check the FiWare common ngsi model and the official OMA documentation as well.
- **INPUT**: QueryContextRequest XML data structure.
- **RETURN**: QueryContextResponse XML data structure.

Example URL

http://localhost/EspR4FastData-3.3.3/ngsi10/queryContext (queried with HTTP Post method)

Tutorial / STEP 15: It is possible to choose the required properties to retrieve. In this example we query them all. You can try to only pick some of them.

"LocalizedParcel0".Query Example Request Payload

```xml
<?xml version="1.0" encoding="UTF-8"?>
<queryContextRequest>
   <entityIdList>
      <entityId type="LocalizedParcel" isPattern="false">
         <id>LocalizedParcel0</id>
      </entityId>
   </entityIdList>
</queryContextRequest>
```
"LocalizedParcel0" Query Response Payload

<queryContextResponse>
  <contextResponseList>
    <contextElementResponse>
      <contextElement>
        <entityId isPattern="false" type="LocalizedParcel">
          <id>LocalizedParcel0</id>
        </entityId>
        <attributeDomainName/>
        <contextAttributeList>
          <contextAttribute>
            <name>vehicleId</name>
            <type>xs:string</type>
            <contextValue>Van1</contextValue>
            <metadata/>
          </contextAttribute>
          <contextAttribute>
            <name>deliveryStep</name>
          </contextAttribute>
        </contextAttributeList>
      </contextElement>
    </contextElementResponse>
  </contextResponseList>
</queryContextResponse>
Tutorial / STEP 15: We can see we have retrieved the merged resulting stored event, combining the property values from "Van1" and "Parcel0". We are done with this tutorial.

9.3.4.4  Resource ngsi10/subscribeContext

9.3.4.4.1  Http Post Method

- **DESCRIPTION**: Implementation of the OMA-ngsi subscribeContext operation. This operation lets third party applications subscribe to particular entities, and then receive notifications whenever a subscribed entity state (a context) is updated. Please check the FiWare common ngsi model and the official OMA documentation as well.
- **INPUT**: SubscribeContextRequest XML data structure.
- **RETURN**: SubscribeContextResponse XML data structure.

Example URL

http://localhost/EspR4FastData-3.3.3/ngsi10/subscribeContext (queried with HTTP Post method)

"LocalizedParcel0" Subscribe Example Request Payload

```
<subscribeContextRequest>
  <entityIdList>
    <entityId isPattern="false" type="LocalizedParcel">
```

"LocalizedParcel0" Subscribe Response Payload

9.3.4.5 Resource ngsi10/unsubscribeContext

9.3.4.5.1 Http Post Method

- **DESCRIPTION**: Implementation of the OMA-ngsi unsubscribeContext operation. This operation lets third party applications unsubscribe to particular entities, and then stop receiving notifications whenever a subscribed entity state (a context) is updated. Please check the FiWare common ngsi model and the official OMA documentation as well.
- **INPUT**: UnsubscribeContextRequest XML data structure that features a subscription identifier.
- **RETURN**: UnsubscribeContextResponse XML data structure.
Example URL

http://localhost/EspR4FastData-3.3.3/ngsi10/subscribeContext
(queried with HTTP Post method)

Unsubscribe Example Request Payload

<unsubscribeContextRequest>
  <subscriptionId>3b70c1ad-0405-47f0-b761-53db720ff1e4</subscriptionId>
</unsubscribeContextRequest>

Unsubscribe Response Payload

<unsubscribeContextResponse>
  <subscriptionId>3b70c1ad-0405-47f0-b761-53db720ff1e4</subscriptionId>
  <statusCode>
    <code>200</code>
    <reasonPhrase>OK</reasonPhrase>
  </statusCode>
</unsubscribeContextResponse>

9.3.5 Complex Event Processing Features Tutorial

9.3.5.1 REST Resources Overview

- http://[ServerRoot]/EspR4FastData-3.3.3/cep/instance
- http://[ServerRoot]/EspR4FastData-3.3.3/cep/eventTypes
- http://[ServerRoot]/EspR4FastData-3.3.3/cep/eventTypes/{name}
- http://[ServerRoot]/EspR4FastData-3.3.3/cep/statements
- http://[ServerRoot]/EspR4FastData-3.3.3/cep/statements/{name}
- http://[ServerRoot]/EspR4FastData-3.3.3/cep/statements/{name}/eventsinkurls
9.3.5.2  **Resource cep/eventTypes**

9.3.5.2.1  **Http Get Method**

- **DESCRIPTION**: Read and return all configured event types.
- **INPUT**: No payload / No URL parameters.
- **RETURN**: EventTypeList XML data structure.

**Tutorial / STEP 5**: Let's call the following URL to check the configured event types:

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/cep/eventTypes  (queried with HTTP Get method)
```

**Example Request Payload**

No payload

**Response Payload**

```
<eventTypeList>
  <eventType>
    <name>ParcelMonitoring</name>
    <propertyList>
      <property>
        <name>entityIdId</name>
        <type>xs:string</type>
      </property>
      <property>
        <name>attributeDomainName</name>
        <type>xs:string</type>
      </property>
    </propertyList>
  </eventType>
</eventTypeList>
```
<property>
    <name>vehicleId</name>
    <type>xs:string</type>
</property>

<property>
    <name>deliveryStep</name>
    <type>xs:string</type>
</property>

<property>
    <name>metadata_timestamp</name>
    <type>xs:string</type>
</property>

</propertyList>
</eventType>

<eventType>
    <name>VehicleLocation</name>
    <propertyList>
        <property>
            <name>entityIdId</name>
            <type>xs:string</type>
        </property>
        <property>
            <name>attributeDomainName</name>
            <type>xs:string</type>
        </property>
    </propertyList>
</eventType>
Tutorial / STEP 5: We can see that the two previous "registerContext" have been translated by the application into something simpler and clearer. Not only the "VehicleLocation" and "ParcelMonitoring" types are now recognized as ngsi types, but they are also known as event types by the CEP component, which is now ready to receive events that belong to these types.

Now, let's create a statement that merges the "VehicleLocation" and "ParcelMonitoring" types.

9.3.5.3 Resource cep/statements

9.3.5.3.1 Http Get Method

Tutorial / STEP 11: Let's query the whole statements list thanks to this operation. All configured statement are returned.

- DESCRIPTION: This operation returns all configured available rule statements.
- INPUT: No input payload.
- RETURN: Statement list, including the related event types and the event sinks (recipients).

Example URL
Example Request Payload

No payload.

Response Payload

```xml
<statementList>
  <statement>
    <name>LocalizedParcelStatement</name>
    <epl>INSERT INTO Localized Parcel
    SELECT 'LocalizedParcel0' as entityIdId,
            ParcelMonitoring.vehicleId as vehicleId,
            ParcelMonitoring.deliveryStep as deliveryStep,
            VehicleLocation.latitude as latitude,
            VehicleLocation.longitude as longitude,
            VehicleLocation.metadata_timestamp as metadata_timestamp
    FROM VehicleLocation.win:time(1000),
        ParcelMonitoring.win:time(1000)
    WHERE ParcelMonitoring.vehicleId = VehicleLocation.entityIdId
         AND ParcelMonitoring.metadata_timestamp = VehicleLocation.metadata_timestamp</epl>
  </statement>
</statementList>
```

<propertyList>
  <property>
    <name>entityIdId</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>vehicleId</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>deliveryStep</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>latitude</name>
    <type>xs:double</type>
  </property>
  <property>
    <name>longitude</name>
    <type>xs:double</type>
  </property>
  <property>
    <name>metadata_timestamp</name>
    <type>xs:string</type>
  </property>
</propertyList>
<eventSinkUrlList>
Tutorial / STEP 11: We get again the same statement. If there were many of them, they all would have been returned. It's now time to send some events to EspR4FastData.

9.3.5.4 Resource cep/statements/{name}

9.3.5.4.1 Http Post Method

Tutorial / STEP 6: Given the required event types have been created, we can now write a statement (rule) that merges them.

- **DESCRIPTION**: Create a CEP rule statement.
- **INPUT**: Statement XML data structure that features the rule statement to create.
- **RETURN**: "Statement" XML data structure.

Example URL

http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement (queried with HTTP Post method)

Example Request Payload
<?xml version="1.0" encoding="UTF-8"?>
<statement>
    <epl>
        INSERT INTO LocalizedParcel
        SELECT 'LocalizedParcel0' as entityIdId,
            ParcelMonitoring.vehicleId as vehicleId,
            ParcelMonitoring.deliveryStep as deliveryStep,
            VehicleLocation.latitude as latitude,
            VehicleLocation.longitude as longitude,
            VehicleLocation.metadata_timestamp as metadata_timestamp
        FROM VehicleLocation.win:time(10),
            ParcelMonitoring.win:time(10)
        WHERE ParcelMonitoring.vehicleId = VehicleLocation.entityIdId
        AND
            ParcelMonitoring.metadata_timestamp = VehicleLocation.metadata_timestamp
    </epl>
</statement>

We can see that the syntax really resembles SQL. The join condition is based on the timestamp, because we want to merge two events which are supposed to occur at the same time. Creating this statement also implies the spontaneous generation of a new event type, which is "LocalizedParcel".

Response Payload

<statement>
<name>LocalizedParcelStatement</name>

<epl>INSERT INTO LocalizedParcel

SELECT 'LocalizedParcel0' as entityIdId,
       ParcelMonitoring.vehicleId as vehicleId,
       ParcelMonitoring.deliveryStep as deliveryStep,
       VehicleLocation.latitude as latitude,
       VehicleLocation.longitude as longitude,
       VehicleLocation.metadata_timestamp as metadata_timestamp

FROM VehicleLocation.win:time(1000),
     ParcelMonitoring.win:time(1000)

WHERE ParcelMonitoring.vehicleId = VehicleLocation.entityIdId

AND ParcelMonitoring.metadata_timestamp = VehicleLocation.metadata_timestamp</epl>

<targetEntityIdType>LocalizedParcel</targetEntityIdType>

<propertyList>
  <property>
    <name>entityIdId</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>vehicleId</name>
    <type>xs:string</type>
  </property>
</propertyList>
Tutorial / STEP 6: The response is quite verbose: the first "propertyList" describes the properties of the new "LocalizedParcel" event type. The "relatedEventTypeList" element lists all the event types that are related to, or used by, the new "LocalizedParcel" event type.

As a STEP 7 activity, you might check that the new "LocalizedParcel" event type has really been created by calling again "http://localhost/EspR4FastData-3.3.3/cep/eventTypes". As the operation uses an HTTP Get method, you can directly click on the link in the browser.

Now, let's move on to the event sink (event recipient) configuration.
9.3.5.4.2 **Http Get Method**

- **DESCRIPTION**: Read and return a particular CEP rule statement.
- **INPUT**: No payload but the statement name is passed within the URL path.
- **RETURN**: "Statement" XML data structure.

**Example URL**

```
http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement (queried with HTTP Get method)
```

**Example Request Payload**

No payload

**Response Payload**

```
<statement>
  <name>LocalizedParcelStatement</name>
  <epl>
    INSERT INTO LocalizedParcel
    SELECT 'LocalizedParcel0' as entityIdId,
            ParcelMonitoring.vehicleId as vehicleId,
            ParcelMonitoring.deliveryStep as deliveryStep,
            VehicleLocation.latitude as latitude,
            VehicleLocation.longitude as longitude,
            VehicleLocation.metadata_timestamp as metadata_timestamp
    FROM VehicleLocation.win:time(10),
            ParcelMonitoring.win:time(10)
    WHERE ParcelMonitoring.vehicleId = VehicleLocation.entityIdId
  </epl>
</statement>
```
AND

ParcelMonitoring.metadata_timestamp = VehicleLocation.metadata_timestamp</epl>

<targetEntityIdType>LocalizedParcel</targetEntityIdType>
<ngsi>true</ngsi>
<propertyList>
  <property>
    <name>entityIdId</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>vehicleId</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>deliveryStep</name>
    <type>xs:string</type>
  </property>
  <property>
    <name>latitude</name>
    <type>xs:double</type>
  </property>
  <property>
    <name>longitude</name>
    <type>xs:double</type>
  </property>
<relatedEventTypeList>
    <relatedEventType>
        <name>ParcelMonitoring</name>
        <propertyList>
            <property>
                <name>entityIdId</name>
                <type>xs:string</type>
            </property>
            <property>
                <name>attributeDomainName</name>
                <type>xs:string</type>
            </property>
            <property>
                <name>vehicleId</name>
                <type>xs:string</type>
            </property>
            <property>
                <name>deliveryStep</name>
                <type>xs:string</type>
            </property>
            <property>
                <name>metadata_timestamp</name>
                <type>xs:string</type>
            </property>
        </propertyList>
    </relatedEventType>
</relatedEventTypeList>
<relatedEventType>
  <name>LocalizedParcel</name>
  <propertyList>
    <property>
      <name>entityIdId</name>
      <type>xs:string</type>
    </property>
    <property>
      <name>vehicleId</name>
      <type>xs:string</type>
    </property>
    <property>
      <name>deliveryStep</name>
      <type>xs:string</type>
    </property>
    <property>
      <name>latitude</name>
      <type>xs:double</type>
    </property>
    <property>
      <name>longitude</name>
      <type>xs:double</type>
    </property>
  </propertyList>
</relatedEventType>
<name>metadata_timestamp</name>
  <type>xs:string</type>
</property>
</propertyList>
</relatedEventType>
<relatedEventType>
  <name>VehicleLocation</name>
  <propertyList>
    <property>
      <name>entityIdId</name>
      <type>xs:string</type>
    </property>
    <property>
      <name>attributeDomainName</name>
      <type>xs:string</type>
    </property>
    <property>
      <name>latitude</name>
      <type>xs:double</type>
    </property>
    <property>
      <name>longitude</name>
      <type>xs:double</type>
    </property>
    <property>
      <name>metadata_timestamp</name>
      <type>xs:string</type>
    </property>
  </propertyList>
</relatedEventType>
9.3.5.4.3  

**Http Delete Method**

- **DESCRIPTION**: Delete a statement.
- **INPUT**: No payload but the statement name is part of the called URL.
- **RETURN**: "StatementList" XML data structure.

**Example URL**

http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement (queried with HTTP Delete method)

**Example Request Payload**

No payload

**Response Payload**

<statementList/>
9.3.5.5  Resource cep/statements/{name}/eventsinkurls

9.3.5.5.1  Http Get Method

Tutorial / STEP 9: Here is the URL to call in order to read all the event sinks that are related to the "LocalizedParcelStatement" rule statement.

- **DESCRIPTION**: Read and return all available event sinks.
- **INPUT**: No payload / No URL parameters.
- **RETURN**: "EventSinkUrlList" XML data structure.

**Example URL**

```plaintext
http://localhost/EspR4FastData-3.3.3/cep/statements/{name}/eventsinkurls (queried with HTTP Get method)
```

**Example Request Payload**

```plaintext
No payload
```

**Response Payload**

```xml
<eventSinkUrlList>
  <eventSinkUrl>
    <name>LocalizedParcelSink</name>
    <httpMethod>POST</httpMethod>
    <target>http://localhost/EventSink/ngsi10</target>
    <registrationURL>http://localhost/EventSink/ngsi9</registrationURL>
    <registered>false</registered>
  </eventSinkUrl>
</eventSinkUrlList>
```

Tutorial / STEP 9: Of course there is only one event sink, which is the one we have configured. But thanks to this operation, one can read all available event sinks at once, when there are many of them. This event sink is also available when querying its parent "LocalizedParcelStatement" rule statement.
As a STEP 10 activity, call the "http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement" URL, to check that the event sink is well configured.

Then move to the next step.

9.3.5.6 Resource cep/statements/{statementName}/eventsinkurls/{eventSinkUrlName}

9.3.5.6.1 Http Post Method

Tutorial / STEP 8: We are about to configure an event sink. An event sink is a HTTP target for EspR4FastData output events. Event sinks must be ngsi-compliant. When a rule statement is triggered, the related event is fired towards the defined event sinks. Practically speaking, it means sending an ngsi-updateContext payload (seen later in this tutorial) to a target recipient.

Here is how to create an event sink which is named "LocalizedParcelSink", and which is related to the "LocalizedParcelStatement" statement:

- **DESCRIPTION**: Create a new event sink, which is a target recipient URL for output events that are related to a particular CEP rule statement.
- **INPUT**: StatementName + EventSinkName parameters are part of the URL. As payload, an "EventSink" XML data structure.
- **RETURN**: "EventSink" XML data structure.

Example URL

http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement/eventsinkurls/LocalizedParcelSink (queried with HTTP Post method)

Tutorial / STEP 8: We can see below that there are two URL to fill in: the <target> element contains the actual event recipient, and the <ngsi9URL> element contains the ngsi registration URL. These are often the same, but depending on the architecture, these can be different URLs. It works this way: if the "LocalizedParcelStatement" rule is triggered for the first time, then the "LocalizedParcel" type is registered (see the CEP rule statement) to the event registrar, before sending any event to the event sink. Just like EspR4FastData, the event sink is ngsi-compliant, and as such, it normally needs entities-type registration prior to be able to process updateContext payloads.

Example Request Payload

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
<eventSinkUrl>
  <httpMethod>POST</httpMethod>
  <target>http://localhost/EventSink/ngsi10</target>
</eventSinkUrl>

Response Payload

<eventSinkUrl>
  <name>LocalizedParcelSink</name>
  <httpMethod>POST</httpMethod>
  <target>http://localhost/EventSink/ngsi10</target>
  <registrationURL>http://localhost/EventSink/ngsi9</registrationURL>
  <registered>false</registered>
</eventSinkUrl>

Tutorial / STEP 8: the response simply mirrors the request. Let's now query the event sink that we have just configured, to check if it is well configured.

9.3.5.6.2 Http Get Method

- DESCRIPTION: Read and return an event sink.
- INPUT: No payload, but related CEP rule statement and event sink name are part of the URL.
- OUTPUT: "EventSinkUrl" XML data structure.

Example URL

http://localhost/EspR4FastData-3.3.3/cep/statements/LocalizedParcelStatement/eventsinkurls/LocalizedParcelSink (queried with HTTP Get method)

Example Request Payload
No payload

Response Payload

<eventSinkUrl>
    <name>LocalizedParcelSink</name>
    <httpMethod>Post</httpMethod>
    <target>http://localhost/EventSink/ngsi10</target>
    <ngsi9URL>http://localhost/EventSink/ngsi9</ngsi9URL>
    <ngsiRegistered>false</ngsiRegistered>
</eventSinkUrl>

9.3.6  ngsi extensions

9.3.6.1  REST Ressources Overview

- http://[ServerRoot]/EspR4FastData-3.3.3/NgsiExtensions/contextElements
- http://[ServerRoot]/EspR4FastData-3.3.3/NgsiExtensions/contextElements/{timeStamp}

9.3.6.2  Resource NgsiExtensions/contextElements

9.3.6.2.1  Http Get Method

- DESCRIPTION: This operation is not compliant with the ngsi standard. ngsi requires the storage of the last known state of a given context, whereas this method returns all historically stored context elements. This lets one rebuilds all context history across the time.
- INPUT: No payload / No URL parameters.
- RETURN: A list of context elements.

Example URL

http://localhost/EspR4FastData-3.3.3/EspR4FastData-3.3.3/NgsiExtensions/contextElements (queried with HTTP Get method)
**Example Payload**

No payload.

**Example XML Response**

```xml
<contextRegistrationList>
  <contextRegistration>
    <RegistrationId>465a7021-bf5c-4070-a282-5dd913eaf1b9</RegistrationId>
    <entityIdList>
      <entityId isPattern="false" type="ParcelMonitoring">
        <id>Parcel0</id>
      </entityId>
    </entityIdList>
    <contextRegistrationAttributeList>
      <contextRegistrationAttribute>
        <name>vehicleId</name>
        <type>xs:string</type>
        <isDomain>false</isDomain>
      </contextRegistrationAttribute>
      <contextRegistrationAttribute>
        <name>deliveryStep</name>
        <type>xs:string</type>
        <isDomain>false</isDomain>
      </contextRegistrationAttribute>
    </contextRegistrationAttributeList>
    <registrationMetaData>
      <contextMetadata>
      </contextMetadata>
    </registrationMetaData>
  </contextRegistration>
</contextRegistrationList>
```
<name>timestamp</name>
  <type>xs:dateTime</type>
  <value/>
</contextMetadata>
</registrationMetaData>
<providingApplication>http://localhost/EspR4FastData-3.3.3</providingApplication>
<duration>P5D</duration>
</contextRegistration>
<contextRegistration>
  <RegistrationId>d56a0e9e-a599-4258-a689-8a6f6ec8de79</RegistrationId>
  <entityIdList>
    <entityId isPattern="false" type="VehicleLocation">
      <id>Plane</id>
    </entityId>
    <entityId isPattern="false" type="VehicleLocation">
      <id>Van1</id>
    </entityId>
    <entityId isPattern="false" type="VehicleLocation">
      <id>Van2</id>
    </entityId>
  </entityIdList>
  <contextRegistrationAttributeList>
    <contextRegistrationAttribute>
      <name>latitude</name>
      <type>xs:double</type>
      <isDomain>false</isDomain>
    </contextRegistrationAttribute>
  </contextRegistrationAttributeList>
</contextRegistration>
9.3.6.3  

**Resource NgsiExtensions/contextElements/{timeStamp}**

9.3.6.3.1  

**Http Get Method**

- **DESCRIPTION:** This operation is not compliant with the ngsi standard. ngsi requires the storage of the last known state of a given context, whereas this method returns all historically stored context elements from a given integer timestamp. This lets one rebuilds all context history across the time.

- **INPUT:** No payload / No URL parameters.

- **RETURN:** A list of context elements.

**Example URL**
Future Internet Core Platform

Example Payload

No payload.

Example XML Response

```xml
<contextElementList>
  <contextElement>
    <gmtMsTimeStamp>1396276854173</gmtMsTimeStamp>
    <entityId isPattern="false" type="LocalizedParcel">
      <id>LocalizedParcel0</id>
    </entityId>
    <contextAttributeList>
      <contextAttribute>
        <name>vehicleId</name>
        <type>xs:string</type>
        <contextValue>Van1</contextValue>
      </contextAttribute>
      <contextAttribute>
        <name>deliveryStep</name>
        <type>xs:string</type>
        <contextValue>DeliveryInProgress</contextValue>
      </contextAttribute>
      <contextAttribute>
        <name>latitude</name>
        <type>xs:string</type>
        <contextValue>40.7128</contextValue>
      </contextAttribute>
    </contextAttributeList>
  </contextElement>
</contextElementList>
```
9.3.7 Advanced topics

9.3.7.1 Sliding Window And 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:

```sql
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:

9.3.7.2 **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.
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:

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

9.3.7.3 **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.
```
select * from Withdrawal.win:length(5)
where amount >= 200
```
10 Template Handler - Template Handler - User and Programmers Guide

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

10.1 Introduction

Template Handler is an implementation of its corresponding Open Specification.

This guide shows how to model BPMN processes augmented with new FI-WARE IoT BPMN specific elements and execute them on the execution platform. Template Handler a fully integrated toolkit so users can use BPMN and NGSI together on a high level of abstraction. All user interaction happens based on the graphical web interface.

10.1.1 What does Template Handler offer?

In general, Template Handler is a modeling and execution platform for BPMN processes. In order to be integrated in a real world environment, Template Handler enables users to access information on existing entities. The transmission is based on the NGSI standard. However, Template Handler acts on a very high level of abstraction. Users without any detailed knowledge of the NGSI protocol fundament in the background are able to use measured information about these real world entities within business processes. The BPMN standard is extended by the following new elements:

- **PhysicalEntity**: A Physical Entity in an IoT-aware business process represents the real-world entity, with which the process interacts. Its graphical representation is a collapsed pool decorated with a cow. In order to specify the real-world object, an NGSI EntityId has to be defined by the process modeler. Its symbol is the picture below, a box with a cow picture and the entity's id as label.

  ![Cow](image)

- **SensingTask**: A Sensing Task is a newly introduced BPMN task type needed for the interaction of the process with the Physical Entity. It is drawn as a BPMN task decorated with a WLAN symbol. For the execution of such a task, a NGSI10 queryContext operation is executed. Therefore the ContextAttribute to be measured must be specified in the SensingTask by the process modeler. The EntityId for the NGSI queryContext operation is taken from the PhysicalEntity to which the SensingTask is connected. After the execution of the queryContext operation the value returned from the NGSI Context Management Component is written to the DataObject connected to the SensingTask, so that it is available in during the remainder of the execution of the
process. Its symbol is the one under the text, a task symbol with a start event in the upper left corner.

- **NGSIStartEvent**: If a business process should be started due to certain events in the real world, an NGSIStartEvent can be used as the first element of a sequence flow. Like the SensingTask it is connected to a PhysicalEntity, which hold the specification of the NGSI EntityId, and a DataObject, which receives the real-world attribute value. When a business process with an NGSIStartEvent is loaded into a business process execution engine, the engine sends a subscribeContextRequest to the NGSI Context Management Component. Therefore the attributes to be observed for events, the NGSI NotifyCondition and the duration of the subscription have to be specified in the NGSIStartEvent by the process modeler. When the execution engine receives the notifyContextRequest from the NGSI Context Management Component, it writes the real-world value into the associated DataObject and starts the process. Its symbol is the following:

10.2 User's guide

10.2.1 Modeling

10.2.1.1 **Getting started**

When you open the home page of the modeler within your template handler instance you get an overview screen as seen below.
The modeler page includes an overview of existing models as well as the possibility to create new models. To open an existing model just double-click on it. To create a new model click on new - Business Process Model Diagram. In both cases an editor window will open. Elements from the standard BPMN catalogue appear on the left-hand side. You can drag'n'drop them onto the drawing board and create business processes. By saving them you can select a name for them and click on the save button. To access the new Fi-WARE IoT BPMN click on the arrow next to "BPMN (Core Elements)" and select "NGSI elements" from the list.

### 10.2.1.1.1 Physical entities

Sensing tasks as well as NGSI start events both refer to real world entities. These physical entities have their own separate representation in the business model. They are added the following way: In the modeler, select the physical entity from the NGSI elements. Place it in your model. You can double-click on the element to assign a display name. Click attributes on the right-hand side. There you can specify the actual unique entity id according to the NGSI specification.

### 10.2.1.1.2 Sensing Tasks

To measure certain values you can use a sensing task. It queries values from the NGSI server and writes them to a connected data object. From there, the values can later be read by the business process. Two values have to be specified: You can determine which context attributes of which physical entity are read. In order to define the physical entity just create an element for it as described in the previous section. Connect it with a message flow to the task. Having selected "BPMN (Complete)" you can find the "Data Object" under "Data Object". From "Connecting Objects" create an "Association (unidirectional)" from the task to the data object. Finally you can the context attribute to be measured in the task's attribute view on the right-hand side. Under "Context Attribute" enter its name.

### 10.2.1.1.3 NGSI start events

A NGSI start event is triggered whenever certain conditions are fulfilled. The proceeding sequence will be executed and the data are stored in a data object in order to be used in the process. You can select a physical entity and a data sink in the same manner as for the sensing
task. In contrast to the changing task you can put multiple select multiple context attributes whose values shall be written to the data object. Put them under "Transmitted context attributes" and separate them by line breaks. The condition triggering the start event can be set the following way. The pictures below show a valid configuration for each type of event triggering condition: Under "Notify condition" select one of the following values:

- **NONE**: Whenever the context attributes are updated on the server, this event is triggered

- **ONTIMEINTERVAL**: The event is triggered regularly. If this is selected, also select the "Time interval" in seconds.
ONCHANGE: When certain context attribute values of the observed entity change this event is triggered. To select the relevant context attributes, enter their names line-break separated for "Triggering context values"
10.2.2 Hello World example

Navigate to the start page of your instance's modeler. The Installation Guide gives more information on how to find the URL.

If you do not have an instance yet, you can use the Test Bed instance.

Select "New" and afterwards "Business Process Model".

In the freshly opened editor, select "NGSI" for the Combo Box above the element list and drag’n’drop a PhysicalEntity to the Canvas.

Left click the PhysicalEntity and have a look at the attribute list on the right-hand side. You can now assign an entity ID to the PhysicalEntity.
Click the "Save" button in the upper left corner and name the file HelloWorld.

Now navigate to the directory where the models are stored. Please have a look to the Installation guide on how to install it or visit [1] for a list from the Test Bed instance's models.

Open HelloWorld.bmpn20.xml with an editor. You can see the placed PhysicalEntity with its correct id.
10.3 Programmer's guide

10.3.1 NGSI server operations

The NGSI server implements a subset of the operations specified by the NGSI-10 Open RESTful binding.

The following operations are supported:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Base URI:</th>
<th>HTTP verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>http://{serverRoot}/NGSI10</td>
<td><strong>POST</strong></td>
</tr>
<tr>
<td>Context query resource</td>
<td>/queryContext</td>
<td>Generic queries for context information. The expected request body is an instance of queryContextRequest; the response body is an instance of queryContextResponse.</td>
</tr>
<tr>
<td>Subscribe context resource</td>
<td>/subscribeContext</td>
<td>Generic subscriptions for context information. The expected request body is an instance of subscribeContextRequest; the response body is an instance of subscribeContextResponse.</td>
</tr>
<tr>
<td>Update context resource</td>
<td>/updateContext</td>
<td>Generic context updates. The expected request body is an instance of updateContextRequest; the response body is an instance of updateContextResponse.</td>
</tr>
</tbody>
</table>

In order to notify clients after they used subscribeContext, the outgoing notification follows the notifyContext operation:

<table>
<thead>
<tr>
<th>Resource</th>
<th>URI</th>
<th>HTTP verbs</th>
</tr>
</thead>
</table>

For a more complete example, please have a look at [Test 2 from the Unit Testing Plan](#).
### 10.3.2 The background processing

In order to use the new NGSI components in production a comprehension of the processing mechanisms is not required. Due to simplicity and accessibility being part of the Template Handler’s concept, it’s abstracted from the technical details. The idea is to enable users to describe the NGSI start event in a graphical interface intuitively.

However, when the infrastructure of the Template Handler is deployed it might be useful to know the underlying concepts. This helps for example to configure the network or exchange tools in the running system.

The diagram below shows the communication sequences when a user models and executes a business process containing a NGSI start event.

<table>
<thead>
<tr>
<th>Notify context resource</th>
<th>Generic notification. The expected request body is an instance of notifyContextRequest; the response body is an instance of notifyContextResponse.</th>
</tr>
</thead>
</table>