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

D.3.4.3: 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-WP3-D.3.4.3  
**Project Document Date:** 2014-04-30  
**Deliverable Type and Security:** Public  
**Author:** FI-WARE Consortium  
**Contributors:** FI-WARE Consortium
1.1 Executive Summary

This document describes the usage of each Generic Enabler provided by the "Application and Services Ecosystem and Delivery Framework" chapter. Due to the different nature of the Apps and Services GEs, this document takes separate approaches for describing end user facing and backend Generic Enablers. For the backend Generic Enabler Registry and W-Store the necessary steps to develop a software application or a user interface which makes use of the Generic Enablers backend functionality are described. For the end user facing Application Mashup, Light Semantic Composition, Business Modeler, Business Calculator and Revenue Sharing System GEs, a more tutorial-like approach was chosen to guide a user through the process of using GE functionality.
1.2 About This Document

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

1.3 Intended Audience

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

1.4 Chapter Context

The Generic Enablers for the Apps Chapter together can be used to build the core infrastructure for enabling a sustainable ecosystem of applications and services of future internet application domains, which foster innovation as well as cross-fertilization. In particular the Apps Generic Enablers supports unified description and publishing of services, offering of services in a store, matching demand and offering via marketplace capabilities, creating composed value added services and service networks, and monetization and revenue sharing, all in a complementary and harmonized business framework.

The concept of the Generic Enabler implies that there can be several possible implementations. There are various degrees of flexibility in the non-functional properties or functional profile of the Generic Enabler description. Not every GE has a RESTful Web interface. Especially the composition editors expose their functionality mainly through a User Interface. This case requires the interface to be described in an abstract way (e.g. what a user can do) and illustrated by screenshots of specific enabler implementations.

A number of basic enablers are important to realize the vision of such a service business framework which enables new business models in an agile and flexible way:

- **Repository** - defines a standard way of publishing service description in the Web in a scalable way.
- **Registry** - serves as a common database layer for run-time configuration and defines a common model and access interface.
- **Store** - allows to offer services for consumers as well as developers of future internet applications.
- **Marketplace** - defines a standard way to access market places in order to find and compare offerings from different stores and provides further functionality to foster the market for future internet applications and services in a specific domain.
- **Revenue Sharing System** - provides a common scheme and protocols for the calculation and distribution of revenues according to the agreed business models.
- **Composition** - to allow or to perform light semantic composition, furthermore composition of existing services to value added composite services and applications, which can be monetized in the Business Framework.
- **Business Modeler and Business Calculator** - handle the monetization of services or applications as well as their compositions/aggregations. Pricing schemes are modelled in business model definitions. The information in these business model definitions is stored and handled in the rating/charging/billing systems.
- **Mediator** - enables the interoperability between future internet services and applications and also allow to interface to existing enterprise systems.

This set of self-contained enablers represents only an initial starting point for a future business framework. It is expected that supplemental enablers (e.g. for contracting, quotation ...) will be developed outside the FI-WARE projects.

The Business Framework has been designed to interoperate with each other relying on Linked USDL as common uniform description format for services, which does not only focus on technical
aspects of service but also covers business aspects as well as functional and non-functional service attributes. Linked USDL itself is not a Generic Enabler, since it is a data format and vocabulary specification. Nevertheless, it will be introduced as an Open Specification, which is used by different enablers in their provided and consumed APIs.

The Applications and Services Generic Enablers are named according to their main functionality. While the role names, introduced in the FI-WARE Vision (Aggregator, Gateway ...), are used to describe the stakeholders of the service ecosystem in an abstract way, the enablers names now are referring to concrete software components.

The following diagram gives an example of how the Generic Enablers can be combined to form a concrete architecture for a Service Business Framework.

![Diagram of Generic Enablers](http://wiki.fi-ware.org/Architecture_of_Applications_and_Services_Ecosystem_and_Delivery_Framework)

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

- [Architecture of Applications and Services Ecosystem and Delivery Framework](http://wiki.fi-ware.org/Architecture_of_Applications_and_Services_Ecosystem_and_Delivery_Framework)
- [Materializing Applications/Services Ecosystem and Delivery Framework in FI-WARE](http://wiki.fi-ware.org/Materializing_Applications/Services_Ecosystem_and_Delivery_Framework_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.org/](http://wiki.fi-ware.org/)

The following resources were used to generate this document:

- Application Mashup - Wirecloud - User and Programmer Guide
- Light Semantic Composition - User and Programmer Guide
- Marketplace - User and Programmer Guide
- Mediator - User and Programmer Guide
- Registry - User and Programmer Guide
- Repository - User and Programmer 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.
Future Internet Core Platform

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 Version</td>
<td>2014-06-20</td>
<td>E-IIS</td>
</tr>
<tr>
<td>v2</td>
<td>Second Version</td>
<td>2014-06-27</td>
<td>E-IIS</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 .......................................................... 4
1.6 Typographical Conventions .......................................................... 5
  1.6.1 Links within this document ...................................................... 5
  1.6.2 Figures ....................................................................................... 5
  1.6.3 Sample software code ............................................................... 5
1.7 Acknowledgements .......................................................................... 5
1.8 Keyword list .................................................................................... 6
1.9 Changes History ............................................................................. 6
1.10 Table of Contents .......................................................................... 6

2 Application Mashup - Wirecloud - User and Programmer Guide ................. 10
  2.1 Introduction .................................................................................. 10
  2.2 User Guide ................................................................................... 10
    2.2.1 Key Features ........................................................................... 10
    2.2.2 Creating a new workspace ...................................................... 11
    2.2.3 Browsing the Marketplace ...................................................... 13
    2.2.4 Building a new mashup ......................................................... 20
    2.2.5 Sharing your mashups ............................................................ 36
    2.2.6 Additional sources of information ......................................... 38
  2.3 Programmer Guide ......................................................................... 39

D.3.4.3: FI-WARE User and Programmer Guide 6
2.3.1 Widget and Operator development ................................................................. 39
2.3.2 Additional sources of information ................................................................. 74
3 Light Semantic Composition - User and Programmer Guide ............................... 75
  3.1 Introduction ........................................................................................................ 75
  3.2 User Guide ......................................................................................................... 75
    3.2.1 Login to GE................................................................................................. 75
    3.2.2 Access to the Editor ................................................................................... 80
    3.2.3 Composition Editor ..................................................................................... 82
    3.2.4 Discover services through semantic annotation ......................................... 82
    3.2.5 Service discovery through the Marketplace GE ......................................... 85
    3.2.6 Services List Widget .................................................................................... 90
    3.2.7 Save the process .......................................................................................... 91
    3.2.8 Generate BPMN 2.0 file ............................................................................. 92
    3.2.9 Deploy and execute the BPMN 2.0 file ....................................................... 95
    3.2.10 Limitations and recommendations ............................................................. 100
  3.3 Programmer guide ............................................................................................. 100
4 Marketplace - User and Programmer Guide ......................................................... 101
  4.1 Introduction ........................................................................................................ 101
  4.2 User Guide ......................................................................................................... 101
    4.2.1 Using the Pricing Support Component ....................................................... 101
  4.3 Programmer Guide ............................................................................................. 106
    4.3.1 Accessing the Marketplace with cURL ....................................................... 107
    4.3.2 Accessing the Marketplace with Java ......................................................... 108
    4.3.3 Using the Review and Rating Component of the Marketplace ................... 109
    4.3.4 Using the Rating based Recommendation Component of the Marketplace .. 113
5 Mediator - User and Programmer Guide ............................................................... 114
  5.1 Introduction ........................................................................................................ 114
  5.2 User Guide ......................................................................................................... 114
  5.3 Programmer Guide ............................................................................................. 114
    5.3.1 Camel Routes Handling .............................................................................. 114
    5.3.2 User Logs .................................................................................................... 115
    5.3.3 Examples ..................................................................................................... 115
    5.3.4 Back-end services exposed ......................................................................... 128
6 Registry - User and Programmer Guide ................................................................. 129
  6.1 Introduction ........................................................................................................ 129
  6.2 User Guide ......................................................................................................... 129
  6.3 Programmer Guide ............................................................................................. 129
    6.3.1 Accessing the Registry with cURL ............................................................. 129

D.3.4.3: FIWARE User and Programmer Guide 7
# Future Internet Core Platform

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Repository - User and Programmer Guide</td>
<td>133</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>133</td>
</tr>
<tr>
<td>7.2</td>
<td>User Guide</td>
<td>133</td>
</tr>
<tr>
<td>7.3</td>
<td>Programmer Guide</td>
<td>133</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Accessing the Repository with cURL</td>
<td>133</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Accessing the Repository with Java</td>
<td>135</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Retrieving Meta Data information about a Resource in different formats</td>
<td>136</td>
</tr>
<tr>
<td>8</td>
<td>RSS - User and Programmer Guide</td>
<td>147</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>147</td>
</tr>
<tr>
<td>8.2</td>
<td>User Guide</td>
<td>147</td>
</tr>
<tr>
<td>8.2.1</td>
<td>RSS Back End Interface</td>
<td>147</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Set Up new Store</td>
<td>154</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Pentaho BI Server Interface</td>
<td>154</td>
</tr>
<tr>
<td>8.3</td>
<td>Programmer Guide</td>
<td>162</td>
</tr>
<tr>
<td>8.3.1</td>
<td>RSS API REST service</td>
<td>162</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Balance and Accumulated API REST service</td>
<td>169</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Expenditure limit management API REST service</td>
<td>176</td>
</tr>
<tr>
<td>9</td>
<td>Store - W-Store - User and Programmer Guide</td>
<td>188</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>188</td>
</tr>
<tr>
<td>9.2</td>
<td>User Guide</td>
<td>188</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Profile configuration</td>
<td>188</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Service provider</td>
<td>191</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Customer</td>
<td>213</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Admin</td>
<td>225</td>
</tr>
<tr>
<td>9.3</td>
<td>Programmer Guide</td>
<td>234</td>
</tr>
<tr>
<td>9.3.1</td>
<td>API Authentication and authorization</td>
<td>235</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Resources management integration</td>
<td>237</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Offerings management integration</td>
<td>239</td>
</tr>
<tr>
<td>9.3.4</td>
<td>Purchases</td>
<td>247</td>
</tr>
<tr>
<td>9.3.5</td>
<td>Accounting and Pay-Per-Use integration</td>
<td>250</td>
</tr>
<tr>
<td>10</td>
<td>Business Modeler - User and Programmer Guide</td>
<td>252</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>252</td>
</tr>
<tr>
<td>10.2</td>
<td>User Guide</td>
<td>252</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Business model meta data</td>
<td>252</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Time Range</td>
<td>254</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Business Elements</td>
<td>255</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Submodels</td>
<td>261</td>
</tr>
<tr>
<td>10.2.5</td>
<td>Menu</td>
<td>266</td>
</tr>
</tbody>
</table>

D.3.4.3: FI-WARE User and Programmer Guide
10.3 Programmer Guide .............................................................................................................. 271
10.3.1 CMD ............................................................................................................................. 271
11 Business Calculator - User and Programmer Guide .......................................................... 275
11.1 Introduction ....................................................................................................................... 275
11.2 User Guide ......................................................................................................................... 275
11.2.1 Goal of the Business Calculator and its different parts .............................................. 275
11.2.2 General characteristics of the editors ......................................................................... 275
11.2.3 The ECMN editor ....................................................................................................... 278
11.2.4 The BPMN editor ........................................................................................................ 279
11.2.5 The Revenue editor ..................................................................................................... 279
11.2.6 The hierarchical network architecture ........................................................................ 280
11.3 Programmer Guide .......................................................................................................... 281
11.3.1 The Business Calculator ............................................................................................ 281
11.3.2 A Limited Repository .................................................................................................. 288
2 Application Mashup - Wirecloud - User and Programmer Guide

2.1 Introduction

This document describes the necessary steps for developing application mashups using the open source WireCloud Mashup Platform. In addition to this, this guide also describes how to develop the mashable application components that allows end users to create those application mashups.

2.1.1.1 Background and Detail

This User and Programmers Guide relates to the Application Mashup GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

2.2 User Guide

Web mashups integrate heterogeneous data, application logic, and UI components (widgets/gadgets) sourced from the Web to create new coherent and value-adding composite applications.

Web mashups are targeted at leveraging the "long tail" of the Web of Services by exploiting rapid development, DIY, and shareability. They typically serve a specific situational (i.e. immediate, short-lived, customised, specific) need, frequently with high potential for reuse. Is this "situational" character which preclude them to be offered as 'off-the-self' functionality by solution providers.

Web mashups can be manually developed using conventional web programming technologies (e.g. see http://programmableweb.com). But this approach fails to take full advantage of the approach. Mashup tools and platforms like WireCloud aim at development paradigms that do not require programming skills and, hence, address end users, thus leveraging the long tail of the Web of Services.

WireCloud builds on cutting-edge end-user development, RIA and semantic technologies to offer a next-generation end-user centred mashup platform aimed at leveraging the long tail of the Internet of Services.

2.2.1 Key Features

WireCloud helps end users to innovate through experimentation by choosing the best suited widgets and prefab mashups (a.k.a. mashup-lets) for your devised mashup from a vast, ever-growing distributed catalogue.

WireCloud offers its main features through two integrated tools:

1. The wiring editor, which allows you to easily connect widgets in a mashup to create a full-fledged dashboard with RIA functionality
2. The piping editor, which allows you to easily connect widgets to back-end services or data sources through an extendable set of operators, including filters, aggregators, adapters, etc.

Besides, WireCloud allows you to easily share your newly created mashup with other colleagues and users. Comment it, tag it and rate it to foster discoverability and shareability. WireCloud helps to build a strong community by commenting, tagging and rating others' widgets, operators and mashups. The platform will also do its best to complement your contribution.
2.2.2 Creating a new workspace

Mashups in WireCloud are built in the context of workspaces. A workspace consists of the set of widgets and operators that can be mashed-up, even spanning multiple tabs. Widgets and operators in a workspace can share data through data flow- or event-based mechanisms.

The workspace in use is shown in the upper area of the screen. It resembles the well-known REST nomenclature. For example, the following screenshot shows a workspace named “Workspace”, pertaining the user “alvaro-arranz” and running in the FI-LAB’s instance of WireCloud, i.e. it is named alvaro-arranz/Workspace.

Near the workspace name there is a button that you can click on to expand the workspace dropdown menu:

Once expanded, the menu shows a list of the already created workspaces (see Workspace, My Multimedia Workspace and IssueTrouble in the figure above) that allows you to quickly switch between them, followed by a list of options:
- **Rename** changes the name of the current workspace
- **Settings** changes the settings of the current workspace
- **Remove** deletes the current workspace
- **New workspace** creates a new workspace
- **Embed** shows info about how to embed current workspace in other web pages
- **Upload to local catalogue** allows you to save the current workspace to the local catalogue for later usage

If you want to create a new workspace named “History Info”, choose “New workspace” in the dropdown menu:

A dialog will pop up requiring a name for the new workspace. Type the desired name and click the accept button:

Once accepted, the name of the new workspace is shown in the upper of the window:
The following screenshot shows the Settings menu, where you can set workspace features:

2.2.3 Browsing the Marketplace

2.2.3.1 Marketplaces and Stores

A mashup tool like WireCloud must support access to a marketplace made up of stores, where people can offer and deal with services made accessible through widgets and operators, like goods, and finally mashup them to create value added services and applications.

On the marketplace you can quickly find and compare widgets and operators, which enable you to attend an industry-ecosystem better than before. Widgets, operators, and even pre-built mashups become tradable goods, which can be offered and acquired on Internet based marketplaces. Partner companies and other users can combine existing services to new services whereby new business models will be incurred and the value added chain is extended.
We differentiate the marketplace from a store. While a store is owned by a store owner who has full control over the specific (limited) widget, operator and mashup portfolio and offerings, a marketplace is a platform for many stores to make their offerings available to a broader audience and enable consumers to search and compare widgets, operators and pre-built mashups and find the store, where to buy. The final business transaction (buying) is done at the store and the whole back office process is handled by the store.

The following figure shows a screenshot of WireCloud where you can see the FI-WARE marketplace and the different stores made available through it.

2.2.3.2 Managing marketplaces

When looking for an offer of widgets, operators and mashups, you first need to choose a marketplace. Use the dropdown menu shown in the workspace path for this purpose.
As shown in the previous screenshot, this menu allows you to choose among the different marketplaces you have access to. To add a new marketplace, you only need to provide its endpoint (URL). You can also delete the current marketplace through the “Delete Marketplace” option in the context menu.

If you have only Local marketplace available, you can add the FI-WARE marketplace using "Add new marketplace", as shown in the following screenshots.

WireCloud offers a built-in local catalogue, which allows you to search among the widgets, operators and mashups currently available for the user. The following figure shows a screenshot of the local catalogue for a user in a given instance of WireCloud.
If you are a widget developer with brand new widgets to share, or you just have downloaded a WireCloud-compliant widget from anywhere, you can easily upload your new widgets to the built-in local catalogue through the "Upload" option in the dropdown menu.
2.2.3.3 **Choosing an available store**

Stores in WireCloud are associated to a specific marketplace. Therefore, to surf a store you first need to choose the FI-WARE marketplace that publishes it. In the following figure, the user uses the dropdown menu to choose FI-LAB’s marketplace:
Once in the FI-WARE marketplace, the store dropdown menu shows all its available stores (CoNWeT, WStore FI-Lab and Another Store). The following figure shows the options available in the stores dropdown menu:

Last, but not least, you can return to surf the entire marketplace and see the global offer at a glance by selecting the "All stores" option.

2.2.3.4 **Publishing mashable application components into Stores**

1. Go to the local catalogue:

2. Open the mashable application component details clicking on it:
3. Click on Publish:

4. Check the marketplace and the Store where the mashable application component is going to be uploaded
5. That's all!! Now you will be able to create new offerings at the selected Store using the uploaded resource.

2.2.4 Building a new mashup

If you followed the instructions documented in the "Creating a new workspace" section, you should have a "History Info" workspace. Let's assume that we start from this point:

Go then to the Marketplace to choose among the widgets available in the catalogue those you want to use in your mashup:

To ensure that you find the required widgets for this example mashup, go to the FI-LAB marketplace and install them. You can also download them using the following URLs:

- CoNWeT_simple-history-module2linear-graph_2.3.2.wgt
- CoNWeT_entity-service_2.3.3.wgt
- CoNWeT_entity2poi_2.3.3.wgt
Once installed, you should be presented with a catalogue of widgets that includes the ones used in this example:

Look for the **Linear Graph** widget and add it to your workspace using the Add button:

The tool automatically changes to the Editor view and presents the selected widget in the design canvas. Now you can move and resize it until you obtain the desired layout:
Return to the Marketplace view and add “Map Viewer” widget. After rearranging it you will be presented with the following view, which shows you the two widgets in the default tab. You can see the tabs used in your workspace at the footer bar, and you can create new tabs to better organize the distribution of the widgets in your mashup.

2.2.4.1 Changing the settings of a widget

Once you have added the desired widgets to your mashup and you have placed and resized them to configure the information dashboard of your choice, you can change their settings. To do so, go
to the upper-right corner of the widget and click the properties icon as shown in the following screen shot

You will then be presented with a dropdown menu with several options.

- **Rename** changes the widget name shown in workspace editor and wiring Editor views
- **Settings** shows a form for changing the settings of the current widget
- **Log** shows a dialog with the log history of the widget
- **Reload** reloads the widget
- **User’s Manual** will open the widget documentation
- **Full Dragboard** maximises the selected widget, so it will take up the full canvas area. This option becomes **Exit Full Dragboard** if the widget is already on "Full Dragboard" mode. In that case, this option will restore the size of the widget to the one it had before maximising it
- **Extract from grid** lifts up the widget from the canvas, allowing you to place it wherever you want on the canvas, even on top of other widgets. This option becomes **Snap to grid** if the widget is currently outside the grid, in this case, this option docks the widget into the grid.

Finally, click on the settings and you will be prompted with a customised dialog for the settings of the widget. In this example, the **Map Viewer** should be provided with initial location, zoom level and mark shadow radius to customise the visualisation.
After configuring the settings, the widget will show the new location, Santander, with the new zoom.

At this time, you have created a mashup with two individual widgets. The Linear Graph widget is empty and need to be wired with something that provides information to draw, and the Map Viewer is a good option to show any kind of "Points of Interest" and allow the user to select them easily.
2.2.4.2 **Wiring widgets and operators**

Once you have chosen the desired widgets, you can wire them to enable their intercommunication and to achieve coordinated behaviour. Widgets and operators in WireCloud, are capable of sending and/or receiving events and data through well-identified ports called endpoints. When you connect two compatible endpoints, the second one (i.e. the input or target endpoint) prepares to receive data flows and/or events coming from the first one (i.e. the output or source endpoint).

### 2.2.4.2.1 Basic wiring concepts

To wire the widgets and add operators to your mashup go to the Wiring view of the tool:

You will then be presented with the set of widgets currently added to the workspace and the set of operators currently available:
One of the most important characteristics that should be intrinsic to the design of widgets is that they must be as generic as possible. For example, it makes much more sense to have a generic Map Viewer widget that can be wired through an operator to any source of information of your choice, than to have a specific one that has hard-coded the source of data. Operators represent the means to achieve this generality, because they represent the means to dynamically associate the widgets with the concrete services or sources of information you want to use in the context of a particular mashup.

In this case, we have an Entity Service operator that is going to provide the data information to the Map Viewer widget. This kind of operators are called piping operators. So we have to add it to the wiring. To do so, drag the operator from the operator list to wiring canvas:
Now, we have the source of information that is going to be presented in the *Map Viewer* widget. So we need to add it to the wiring status following the same process for the operator, but dragging the widget instead of the operator:

The wiring editor comes with a recommendation system for connections. For example, move the pointer to the *Provide entity* endpoint. You will see that the endpoint get highlighted, this means that the recommendation system is searching for compatible endpoints. In this case, there are no compatible endpoints.
This is because the output of the Entity Service cannot be connected directly with the Map Viewer widget. We can use a transform operator to convert the event data provided by the Entity Service operator to the format used by the Map Viewer widget. In this example, the operator that is going to perform this transformation is Entity to Pol:

After adding the operator, we can move the pointer to the Provide entity endpoint to see that now we have a connection recommendation:
So, we can connect it. To do so, push down the mouse button on the *Provide entity* endpoint and drag the arrow to the *Entity* endpoint:
And we can also connect also the PoI endpoint of the Entity To PoI operator to the Insert/Update PoI endpoint on the Map Viewer widget:

If you return to the Editor view, you will see that the map widget has been updated and is showing the Pols obtained from the Entity Service operator.
You can use the Map Viewer moving the viewport, selecting PoI's, etc. But in really, what we have is just the Map Viewer widget connected to a source of data, but using piping and transformation operators that is going to give us a great flexibility.

2.2.4.2.2 Other wiring common tasks

One of the most common operations is the task of getting the connections removed in the wiring. For example, when you make some mistake for some reason, you can fix it by selecting the connection (by clicking on it) and then by clicking the red dot that appears in the middle of it.
Another common task is to change the shape of connections. This can be accomplished by moving the handles that appear when they are selected.
You can also minimize operators with the intention of improving space usage. This can be accomplished using the "Minimize" option that appears in the Widget's menu:
2.2.4.2.3 Cloncluding our example

Continue wiring the rest of the widgets in your mashup following your intuition, the documentation and the contextual help offered by each widget/operator. Anyway, just in case you have difficulties, you can see the final result in the following screenshot:

Operators, like widget has also settings that can be modified using the following steps:
Now you can play with your new workspace.
2.2.5 Sharing your mashups

Workspaces can be made public by means of modifying their settings as follows:

1. Click on the workspace menu button and click Settings:
2. You will be presented with a dialog for editing workspace's settings:

![Workspace settings dialog](image)

After making a workspace public, you will be able to share the workspace URL with other users.

2.2.5.1 **Embedding mashups inside other web pages**

All workspaces can be embedded, but take into account that access rules are the same that applies when using the workspace directly from WireCloud. If you don't make the workspace public, you will require users to be logged in WireCloud and having enough access permission. This make changing sharing settings of the workspace a first step before embedding mashups into other web pages.
You can also obtain the code you have to copy & paste into other web pages following those steps:

1. Click on the workspace menu button and click *Embed*:

   ![Embed Code](image)

2. A new window showing you the code for embedding the mashup. Copy & paste it into your HTML document.

2.2.6 Additional sources of information

See [The WireCloud website](https://www.wirecloud.eu) for more information. You will find helpful resources such as demo videos, a demo "sandbox" where you will have the opportunity to register and use the tool as a service for free. In that demo deployment of the platform, you will be provided with a catalogue of general purpose widgets, operators and prefab mashups that you can use to experiment and to build new added-value mashups.
2.3 Programmer Guide

2.3.1 Widget and Operator development

Before starting the creation of a widget, the developer should be aware of certain design principles of the widgets:

- Widgets are supposed to be small, reusable and user-centric web applications.
- Generic widgets are desirable, but ad-hoc solutions are allowed too if they are quick and cheap enough.
- Widgets should be adapted to real problems.
- Widgets are elements of the front-end layer (View). It's not desirable for widgets to perform back-end layer functions (controller or model) because they can be provided by the platform (persistent state).
- During the development of widgets any technology accepted by web browsers (XHTML, JavaScript, SVG, Flash, applets ...) can be used.

Widgets are formed by three different components:

- Template, which is a declarative description of the widget, and represents its main entry point. This file contains, among other things, references to the rest of resources of a widget.
- Code, composed of HTML, JavaScript and CSS files containing the definition and the behaviour of the widget.
- Static resources, such as images, documentation and other static resources.

Operators are very similar to Widgets. The main difference is that operators don't have a view. This means that operators descriptors don't use an initial HTML document as entry point. Instead, operators define the list of javascript files in their descriptor file. There are three types of operators:

- Data sources operators: Operators that provide information that can be consumed by other widgets/operators. For example, a operator that retrieves some type of information from a web service.
- Data targets operators: Operators that are provided information and use it to do some tasks. For example, a operator that receives some information and push it to a web service.
- Data transformation operators: This type of operators can be very useful since they can transform data in order to make it usable by widgets or operators that expect data structure to be slightly different.

2.3.1.1 Javascript API

The Javascript API allow Widgets and Operators to access the functionalities offered by the Application Mashup GE like widget/operator interconnection, state persistence, access to the cross-domain proxy, ...

2.3.1.1.1  MashupPlatform.http

buildProxyURL

builds a URL suitable for working around the cross-domain problem. It is usually handled using the WireCloud proxy but it also can be handled using the access control request headers if the browser has support for them. If all the needed requirements are meet, this function will return a URL without using the proxy.

MashupPlatform.http.buildProxyURL(url, options)

- **url** is the target URL.
**options** is an object with request options (see the request options section for more details).

**makeRequest**

sends a HTTP request.

```javascript
MashupPlatform.http.makeRequest(url, options)
```

- **url** is the URL to which to send the request.
- **options** is an object with a list of request options (see the request options section for more details).

**request options**

**General options:**

- **contentType** (String): The Content-Type header for your request. If it is not provided, the content-type header will be extrapolated from the value of the postBody option (defaulting to application/x-www-form-urlencoded if the postBody value is a String). Specify this option if you want to force the value of the Content-Type header.
- **encoding** (String; default UTF-8): The encoding for the contents of your request. It is best left as-is, but should weird encoding issues arise, you may have to tweak this.
- **method** (String; default POST): The HTTP method to use for the request.
- **responseType** (String; default: ""): Can be set to change the response type. The valid values for this options are: ", "arraybuffer", "blob" and "text".
- **parameters** (Object): The parameters for the request, which will be encoded into the URL for a GET method, or into the request body when using the PUT and POST methods.
- **postBody** (ArrayBufferView, Blob, Document, String, FormData): The contents to use as request body (usually used in post and put requests, but can be used by any request). If it is not provided, the contents of the parameters option will be used instead. Finally, if there are not parameters, request will not have a body.
- **requestHeaders** (Object): A set of key-value pairs, with properties representing header names.
- **forceProxy** (Boolean; default false): Sends the request through the proxy regardless of the other options passed.
- **context** (Object; default null): The value to be passed as the this parameter to the callbacks. If context is null the this parameter of the callbacks is left intact.

**Callback options:**

- **onSuccess**: Invoked when a request completes and its status code belongs in the 2xy family. This is skipped if a code-specific callback is defined (e.g., on200), and happens before onComplete. Receives the response object as the first argument.
- **onFailure**: Invoked when a request completes and its status code exists but is not in the 2xy family. This is skipped if a code-specific callback is defined (e.g. on403), and happens before onComplete. Receives the response object as the first argument.
- **onXYZ** (with XYZ representing any HTTP status code): Invoked just after the response is complete if the status code is the exact code used in the callback name. Prevents execution of onSuccess and onFailure. Happens before onComplete. Receives the response object as the first argument.
- **onComplete**: Triggered at the very end of a request's life-cycle, after the request completes, status-specific callbacks are called, and possible automatic behaviours are processed. Guaranteed to run regardless of what happened during the request. Receives the response object as the first argument.
- **onException**: Triggered whenever an exception arises running any of the onXYZ, onSuccess, onFailure and onComplete callbacks. Receives the request as the first argument, and the exception object as the second one.

**response object**
• request (Request): The request for the current response.
• status (Number): The status of the response to the request. This is the HTTP result code (for example, status is 200 for a successful request).
• statusText (String): The response string returned by the HTTP server. Unlike status, this includes the entire text of the response message ("200 OK", for example).
• response (ArrayBuffer, Blob, String): The response entity body according to responseType, as an ArrayBuffer, Blob or string. This is null if the request is not complete, was not successful or the responseType option of the requests was "."
• responseText (String): The response to the request as text, or null if the request was unsuccessful or the responseType option of the requests was different to "".

2.3.1.1.2  MashupPlatform.log
Object containing the following constants:
• ERROR: Used for indicating an Error level.
• WARN: Used for indicating a Warning level.
• INFO: Used for indicating an Info level.

2.3.1.1.3  MashupPlatform.prefs

get
Retrieves the value of a preference.

MashupPlatform.prefs.get(key)

• key is the preference to fetch.

registerCallback
Registers a callback for listening preference changes.

MashupPlatform.prefs.registerCallback(callback)

• callback is the callback function that will be called when the preferences of the widget changes.

set
Sets the value of a preference.

MashupPlatform.prefs.set(key, value)

• key is the identifier of the preference.
• value is the new value to use for the preference.

2.3.1.1.4  MashupPlatform.operator
This module is only available for operators.

id
returns the operator id.

MashupPlatform.operator.id

log
writes a message into the WireCloud's log console.

MashupPlatform.operator.log(msg, level)

• msg is the text of the message to log.
Future Internet Core Platform

- **level** is an optional parameter with the level to use for logging the message. (See #MashupPlatform.log for available log levels. default: MashupPlatform.log.INFO).

2.3.1.1.5 *MashupPlatform.widget*

This module is only available for widgets.

**getVariable**

returns a widget variable by its name.

```
MashupPlatform.widget.getVariable(name)
```

- **name** is the name of the variable to retrieve.

**drawAttention**

makes WireCloud notify that the widget needs user's attention.

```
MashupPlatform.widget.drawAttention()
```

**id**

returns the widget id.

```
MashupPlatform.widget.id
```

**log**

writes a message into the WireCloud's log console.

```
MashupPlatform.widget.log(msg, level)
```

- **msg** is the text of the message to log.
- **level** is an optional parameter with the level to use for logging the message. (See #MashupPlatform.log for available log levels. default: MashupPlatform.log.INFO).

2.3.1.1.6 *MashupPlatform.wiring*

**pushEvent**

sends an event through the wiring.

```
MashupPlatform.wiring.pushEvent(outputName, data)
```

- **outputName** is the name of the output endpoint as defined in the WDL.
- **data** is the content of the event.

**registerCallback**

registers a callback for a given input endpoint. If the given endpoint already has registered a callback, it will be replaced by the new one.

```
MashupPlatform.wiring.registerCallback(inputName, callback)
```

- **inputName** is name of the input endpoint as defined in the WDL.
- **callback** is the callback function to use when an event reaches the given input endpoint.

2.3.1.2 **Pub Sub API**

Once the pub sub add-on is installed and activated, widgets and operators declaring the use of the PubSub feature can take advantage of the PubSub functionalities through the MashupApplication.SilboPS object. Currently, the MashupApplication.SilboPS object only exports the PubEndPoint, SubEndPoint and Filter classes defined by the original javascript bindings.
provided by SilboPS. Full documentation of SilboPS is available at https://svn.forge.morfeo-project.org/4caast/trunk/WP6/pubsub/README.md.

2.3.1.2.1 Examples

Widget description using the XML flavour of the WDL

```xml
<?xml version="1.0" encoding="UTF-8"?><Template xmlns="http://wirecloud.conwet.fi.upm.es/ns/template#">
    <Catalog.ResourceDescription>
        <Vendor>CoNWeT</Vendor>
        <Name>tourist-social-comments</Name>
        <DisplayName>Tourist - Social comments</DisplayName>
        <Version>0.27</Version>
        <Author>UPM</Author>
        <Mail>4caast@conwet.es</Mail>
        <Description>Chat widget for commenting about tourist locations. Uses Pub/Sub as communication channel</Description>
        <ImageURI>images/tourist-social.png</ImageURI>
        <Requirements>
            <Feature name="PubSub" />
        </Requirements>
    </Catalog.ResourceDescription>
    <Platform.Wiring>
        <InputEndpoint name="place" type="text" description="publish location" label="Messages Location" friendcode="connect_location" />
    </Platform.Wiring>
    <Platform.Link>
        <XHTML href="ps.html"/>
    </Platform.Link>
    <Platform.Rendering width="9" height="25"/>
</Template>
```

Widget description using the RDF flavour of the WDL

```rdf
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix usdl: <http://www.linked-usdl.org/ns/usdl-core#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix wire: <http://wirecloud.conwet.fi.upm.es/ns/widget#> .
```
<http://wirecloud.conwet.fi.upm.es/ns/widget#CoNWeT/tourist-social-comments/0.27> a wire:Widget;
  dc:creator [ a foaf:Person;
    foaf:name "UPM" ];
  dc:description "Chat widget for commenting about tourist locations. Uses Pub/Sub as communication channel";
  dc:title "tourist-social-comments";
  wire:displayName "Tourist - Social comments";
  wire:hasImageUri <images/tourist-social.png>;
  wire:hasPlatformRendering [ a wire:PlatformRendering;
    wire:renderingHeight "25";
    wire:renderingWidth "9" ];
  wire:hasPlatformWiring [ a wire:PlatformWiring;
    wire:hasInputEndpoint [ a wire:InputEndpoint;
      rdfs:label "Messages Location";
      dc:description "publish location";
      dc:title "place";
      wire:friendcode "connect_location";
      wire:inputActionLabel "None";
      wire:type "text" ] ];
  wire:hasRequirement [ a wire:Feature;
    rdfs:label "PubSub" ];
  usdl:hasProvider [ a <http://purl.org/goodrelations/v1#BusinessEntity>;
    foaf:name "CoNWeT" ];
  usdl:utilizedResource <ps.html>;
  usdl:versionInfo "0.27";
  vcard:addr [ a vcard:Work;
    vcard:email "4caast@conwet.es" ] .
<ps.html> a usdl:Resource;
  wire:codeCacheable "True" .

Publishing

1. Get a PubEndpoint from the SilboPS.

2.

3. var myPubEndPoint = new MashupPlatform.SilboPS.PubEndPoint({
4.   open: function(endpoint) {
5.     // react to onopen.
6.   },
7.   close: function(endpoint) {
8.     // react to onclose.
9.   }
10. })
11. Create an Advertise object.

```javascript
var advertise = new MashupPlatform.SilboPS.Advertise()
    .attribute("number",
        MashupPlatform.SilboPS.Type.LONG)
    .attribute("other",
        MashupPlatform.SilboPS.Type.STRING);
```

12. Advertise it.

13.

14. `myPubEndPoint.advertise(advertise);`

15. Create a Notification object.

16.

17. `var notification = new MashupPlatform.SilboPS.Notification()
    .attribute("number",
        MashupPlatform.SilboPS.Type.LONG, 5)
    .attribute("other",
        MashupPlatform.SilboPS.Type.STRING, "don't care");`

20. Publish it.

21.

22. `myPubEndPoint.publish(notification);`

23. Close the endpoint.

24.

25. `myPubEndPoint.close();`

### Subscribing

1. Get a PubEndpoint from the SilboPS.

2.

3. `var mySubEndPoint = new MashupPlatform.SilboPS.SubEndPoint({
    open: function(endpoint) {
        // react to onopen.
    },
    close: function(endpoint) {
        // react to onclose.
    },
    ...,
    advertise: function(endpoint, advertise) {
        // handle advertise
    },
});`
14. unadvertise: function(endpoint, unadvertise) {
15.     // handle unadvertise
16. },
17. notify: function(endpoint, notification) {
18.     // handle notification
19. }
20.});

21. Subscribe to a filter.

22. var cxtFunc = new MashupPlatform.SilboPS.ContextFunction();
23. var filter = new MashupPlatform.SilboPS.Filter()
24. .constrain("other",
25.     MashupPlatform.SilboPS.Type.STRING).startsWith("don'"
26. .filter()
27.
28. mySubEndPoint.subscribe(filter, cxtFunc);

29. Handle notifications.

30. // Use "notify" handler.
31. // ...
32. notify: function(endpoint, notification) {
33.     // handle notification
34. }
35. //...
36. //...

37. Close the endpoint

38. mySubEndPoint.close();

2.3.1.3 NGSI API

First of all, widgets and operators wishing to use the javascript bindings provided by WireCloud for accessing the Fi-WARE NGSI Open RESTful API in order to seamlessly interoperate with the Orion Pub/Sub Context Broker must add the NGSI feature as a requirement into their description files (config.xml files).

The following is an example of a widget description using the XML flavor of the WDL:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Template xmlns="http://wirecloud.conwet.fi.upm.es/ns/template#">
    <Catalog.ResourceDescription>
        <Vendor>CoNWeT</Vendor>
        <Name>observation-reporter</Name>
```
<Display Name>Observation Reporter</Display Name>
<Author>aarranz</Author>
<Version>1.0</Version>
<Mail>aarranz@conwet.com</Mail>
<Description>Creates a new observation</Description>
<ImageURI>images/catalogue.png</ImageURI>
<iPhoneImageURI>images/smartphone.png</iPhoneImageURI>
<WikiURI>http://www.envirofi.eu/</WikiURI>

<Requirements>
  <Feature name="NGSI"/>
</Requirements>

The RDF flavor of the same widget description is:

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:foaf="http://xmlns.com/foaf/0.1/
  xmlns:wire="http://wirecloud.conwet.fi.upm.es/ns/widget#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:usdl="http://www.linked-usdl.org/ns/usdl-core#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns1="http://purl.org/goodrelations/v1#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:vcard="http://www.w3.org/2006/vcard/ns#">
  <wire:Widget
    rdf:about="http://wirecloud.conwet.fi.upm.es/ns/widget#CoNWeT/observation-reporter/1.0"
    >
    <vcard:addr>
      <vcard:Work rdf:nodeID="Nb17ce611aa2645e488515f86eb855e53">
        <vcard:email>aarranz@conwet.com</vcard:email>
      </vcard:Work>
    </vcard:addr>
  </wire:Widget>
</rdf:RDF>
```
Once the NGSI feature is added to the widget/operator description file, widgets and operators will have access to the NGSI JavaScript object that conforms the core of the API. See the Publish/Subscribe Broker - Orion Context Broker - User and Programmers Guide for more details on the operations that can be invoked using the RESTful API. That guide can be used for reference as each of the Pub/Sub Context Broker operations has an equivalent operation in the JavaScript bindings.

What follows exemplifies the use of this API for updating an entity.
var connection = new NGSI.Connection('url of the Orion Pub/Sub Context Broker instance');
connection.updateAttributes([{
    entity: {
        id: 'iss8',
        type: 'Issue'
    },
    attributes: [{
        name: 'technician',
        contextValue: 'tech1'
    }]
}]), {
    onSuccess: function () {
        // notify success
    },
    onFailure: function () {
        // show error
    }
});

2.3.1.3.1 Data types used by the library
- The **Entity** type is used to reference entities. This type is defined as an object composed of the following fields:
  - id is a string with the id of the entity. Sometimes you will be able to use patterns in this field.
  - isPattern is a boolean indicating whether the id field contains a regular expression pattern. This field is optional.
  - type is the type of the entity. This field is optional.
- The **Attribute** type is used to reference attributes. This type is defined as an object composed of the following fields:
  - name is the name of the attribute.
  - type is the type of the attribute. This field is optional.
- The **Duration** type is used to describe time intervals and defined as a string following the format defined at [http://books.xmlschemata.org/relaxng/ch19-77073.html](http://books.xmlschemata.org/relaxng/ch19-77073.html).
- The **Condition** type is used to declare the condition that will trigger notifications. This type is defined as an object composed of the following fields:
  - type is an String containing 'ONTIMEINTERVAL' or 'ONCHANGE'.
  - values is an Array of String. The meaning of this field depends on the value of the type field:
    - 'ONTIMEINTERVAL': exactly one value SHALL be present and SHALL represent the time interval between notifications (using the Duration type).
    - 'ONCHANGE': this element SHALL contain the name(s) of the Context Attributes to be monitored for changes.
- The **AttributeValue** type is used to assign values to attributes. This type is defined as an object composed of the following fields:
  - name is the name of the attribute.
  - type is the type of the attribute. This field is optional.
contextValue is the value to assign to the attribute.

- The AttributeUpdate type is used to describe a context update. This type is defined as an object composed of the following fields:
  - entity is the entity affected by the update. Type: Entity.
  - attributes is the new values for the attributes of the entity. Type: AttributeValue.

- The AttributeDeletion type is used to describe the deletion of attributes from an entity. This type is defined as an object composed of the following fields:
  - entity is the entity affected by the update. Type: Entity.
  - attributes is the new values for the attributes of the entity. Type: Attribute.

### 2.3.1.3.2 NGSI.Connection

A new NGSI.Connection can be instantiated using the following constructor:

```javascript
NGSI.Connection(url[, options])
```

- url (String): is the url of the Orion Pub/Sub Context Broker instance.
- options (Object; default null): is an object with extra options. This parameter may be null if no extra option is needed. Currently supported options are:
  - ngsi_proxy_url (String; default null): URL of the NGSI proxy used for subscriptions.
  - request_headers (Object; default null): A set of key-value pairs, with properties representing header names. These extra headers will be used when making request to the context broker.
  - use_user_fiware_token (Boolean; default: false): Use current user authentication token retrieved from the IdM system.

All the methods of NGSI.Connection supports an options parameter. This parameter is used, among other things, to pass callbacks functions. This parameter is a JavaScript object containing pairs of key value options. Moreover, all the method of NGSI.Connection support at least the following callbacks:

- onSuccess is called when the request finishes successfully. The parameters passed to this callback depends on the invoked method.
- onFailure is called when the request finish with errors.
- onComplete is called when the request finish regardless of whether the request is successful or not.

#### createRegistration

Register context information (entities and attributes) in the NGSI server.

```javascript
createRegistration(entities, attributes, duration, providingApplication[, options])
```

- entities is the list of Entities that are going to be registered.
- attributes is a list of the Attributes that are going to be assigned to the previous list of entities.
- duration is the Duration for this registration.
- providingApplication is the URI of the application to which this registration belongs to.

The onSuccess callback will receive an object with the following fields:

--registrationId is the final assigned id. This id can be used in the updateRegistration and cancelRegistration methods.
- duration is the final assigned duration for this registration.

#### updateRegistration

Updates a particular registration.

```javascript
updateRegistration(regId, entities, attributes, duration, providingApplication[, options])
```
- `regId` is the id of the registration to update.
- `entities` is the list of Entities that is going to replace the previous established one.
- `attributes` is a list of the Attributes that are going to be assigned to the provided list of entities.
- `duration` is the new Duration for the registration identified by `regId`.
- `providingApplication` is the new value for the providingApplication property of the registration.

The `onSuccess` callback will receive an object with the following fields:
- `registrationId` is the id of the registration.
- `duration` is the final assigned duration for this registration.

**cancelRegistration**

Cancels or deletes a particular registration.

```
cancelRegistration(regId[, options])
```

- `regId` is the id of the registration to cancel.

**discoverAvailability**

Discover context information registrations in the NGSI server.

```
discoverAvailability(entities, attributeNames[, options])
```

- `entities` is the list of Entities that are going to be queried.
- `attributeNames` is the list of attribute names that are going to be queried. This parameter is optional and thus null is a valid value.

**query**

Query for context information.

```
query(entities, attributeNames[, options])
```

- `entities` is the list of Entities to query.
- `attributeNames` is the list of attribute names to query. Use null for retrieving all the attributes.

The query method supports an extra option:

- `flat` (Boolean; default: false): This option is used for simplifying the data structure used for representing the returned data. This simplification relies in making the following assumptions about the returned entry set:
  - given an entity id there is only one value for the entity's type parameter
  - entities don't have attributes called id or type
  - entities have only an attribute with a given name
  - attribute types don't matter or are already known

For example, this is the value of the data parameter passed to the `onSuccess` callback when using the `flat` is false (default value):

```
[
  {
    "entity": {
      "id": "van1",
      "type": "Van"
    }
  }
]```


```json
},
  "attributes": [
    {
      "name": "current_position",
      "type": "coordinates",
      "contextValue": "43.47557, -3.8048315"
    }
  ]
},

{ "entity": {
  "id": "van2",
  "type": "Van"
},
  "attributes": [
    {
      "name": "current_position",
      "type": "coordinates",
      "contextValue": "43.47258, -3.8026643"
    }
  ]
},

{ "entity": {
  "id": "van3",
  "type": "Van"
},
  "attributes": [
    {
      "name": "current_position",
      "type": "coordinates",
      "contextValue": "43.47866, -3.7991238"
    }
  ]
},

{ "entity": {
  "id": "van4",
  "type": "Van"
},
  "attributes": [
    {
      "name": "current_position",
      "type": "coordinates",
      "contextValue": "43.48071, -3.7938443"
    }
  ]
}
```
Whereas this is the value of the data parameter when `flat` is true:

```json
{
  "van1": {
    "id": "van1",
    "type": "Van",
    "current_position": "43.47557, -3.8048315"
  },
  "van2": {
    "id": "van2",
    "type": "Van",
    "current_position": "43.47258, -3.8026643"
  },
  "van3": {
    "id": "van3",
    "type": "Van",
    "current_position": "43.47866, -3.7991238"
  },
  "van4": {
    "id": "van4",
    "type": "Van",
    "current_position": "43.471214, -3.7994885"
  }
}
```

`updateAttributes`

Update context information.

`updateAttributes(update[, options])`
- **update** a list of `AttributeUpdates`.

**addAttributes**

Add/update entity attributes. This operation will create the attribute on those entities where it does not exist. In addition to this, this operation will also create entities if they don’t exist (provided that entities were referenced without using patterns).

```javascript
addAttributes(toAdd[, options])
```

- **toAdd** a list of `AttributeUpdates`.

**deleteAttributes**

Delete attributes form entities.

```javascript
deleteAttributes(toDelete[, options])
```

- **toDelete** a list of `AttributeDeletion`.

**createSubscription**

Subscribe to changes in the context information.

```javascript
createSubscription(entities, attributeName, duration, throttling, conditions, options)
```

- **entities** is the list of `Entities` to query in this subscription.
- **attributeNames** is the list of attribute names to query in this subscription.
- **duration** is the `Duration` of this subscription.
- **throttling** is the proposed minimum interval between notifications. This value must be provided using the `Duration` type. `null` is also valid.
- **conditions** is a list of `Conditions` that will trigger queries using the provided information and their subsequent notifications to the `onNotify` callback.

This method, supports a new type of callback: `onNotify`. This callback is required and can be either an URL or a function. In the later case, the NGSI Connection must be created using a NGSI proxy and will be called every time a notification comes from the NGSI server.

The first parameter of a `onNotify` callback function will be an object with the response data.

**updateSubscription**

Update context subscription.

```javascript
updateSubscription(subId, duration, throttling, conditions[, options])
```

- **subId** is the id of the context subscription to cancel.
- **duration** is the `Duration` of this subscription.
- **throttling** is the proposed minimum interval between notifications. This value must be provided using the `Duration` type. `null` is also valid.
- **conditions** is a list of `Conditions` that will trigger queries using the provided information and their subsequent notifications to the `onNotify` callback.

**cancelSubscription**

Cancels or deletes context subscriptions. A subscription cannot be updated after being cancelled.

```javascript
cancelSubscription(subId[, options])
```

- **subId** is the id of the context subscription to cancel.
2.3.1.4 ObjectStorage API

First of all, widgets and operators wishing to use the javascript bindings provided by WireCloud for accessing the FIWARE Object Storage Open RESTful API in order to seamlessly interoperate with the Object Storage must add the ObjectStorage feature as a requirement into their description files (config.xml files).

The following is an example of a widget description using the XML flavour of the WDL:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Template xmlns="http://wirecloud.conwet.fi.upm.es/ns/template#">
  <Catalog.ResourceDescription>
    <Vendor>CoNWeT</Vendor>
    <Name>observation-reporter</Name>
    <Display Name>Observation Reporter</Display Name>
    <Author>aarranz</Author>
    <Version>1.0</Version>
    <Mail>aarranz@conwet.com</Mail>
    <Description>Creates a new observation</Description>
    <Image URI>images/catalogue.png</Image URI>
    <iPhone Image URI>images/smartphone.png</iPhone Image URI>
    <Wiki URI>http://www.envirofi.eu/</Wiki URI>
    <Requirements>
      <Feature name="ObjectStorage"/>
    </Requirements>
  </Catalog.ResourceDescription>
  <Platform.Link>
    <XHTML href="index.html" content-type="text/html" cacheable="true" use-platform-style="true"/>
  </Platform.Link>
  <Platform.Rendering width="5" height="20"/>
</Template>
```

The RDF flavor of the same widget description is:

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:wire="http://wirecloud.conwet.fi.upm.es/ns/widget#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:usdl="http://www.linked-usdl.org/ns/usdl-core#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns1="http://purl.org/goodrelations/v1#"
  xmlns:dcterms="http://purl.org/dc/terms/"
```
<wire:Widget rdf:about="http://wirecloud.conwet.fi.upm.es/ns/widget#CoNWeT/observation-reporter/1.0">
  <vcard:addr>
    <vcard:Work rdf:nodeID="Nb17ce611aa2645e488515f86eb855e53">
      <vcard:email>aarranz@conwet.com</vcard:email>
    </vcard:Work>
  </vcard:addr>
  <usdl:utilizedResource>
    <usdl:Resource rdf:about="index.html">
      <wire:codeCacheable>True</wire:codeCacheable>
    </usdl:Resource>
  </usdl:utilizedResource>
  <wire:hasPlatformWiring>
    <wire:PlatformWiring rdf:nodeID="Neecc97db81ed40859b8c04e935a9a9cc"/>
  </wire:hasPlatformWiring>
  <wire:displayName>Observation Reporter</wire:displayName>
  <wire:hasiPhoneImageUri rdf:resource="images/smartphone.png"/>
  <usdl:versionInfo>1.0</usdl:versionInfo>
  <usdl:hasProvider>
    <ns1:BusinessEntity rdf:nodeID="N9a6bf56577c741ac806997a80281afff">
      <foaf:name>CoNWeT</foaf:name>
    </ns1:BusinessEntity>
  </usdl:hasProvider>
  <wire:hasImageUri rdf:resource="images/catalogue.png"/>
  <wire:hasPlatformRendering>
    <wire:PlatformRendering rdf:nodeID="N713e5e11dce4750a592c754c748def7">
      <wire:renderingHeight>20</wire:renderingHeight>
      <wire:renderingWidth>5</wire:renderingWidth>
    </wire:PlatformRendering>
  </wire:hasPlatformRendering>
  <wire:hasRequirement>
    <wire:Feature rdf:nodeID="N3cb336bd9b6243ecbf345c80442498f9">
      <rdfs:label>ObjectStorage</rdfs:label>
    </wire:Feature>
  </wire:hasRequirement>
</wire:Widget>
Once the ObjectStorage feature is added to the widget/operator description file, widgets and operators will have access to the ObjectStorageAPI javascript object that conforms the core of the API. See the Object Storage - User and Programmers Guide for more details on the operations that can be invoked using the RESTful API. That guide can be used for reference as each of the Object Storage operations have an equivalent operation in the JavaScript bindings.

What follows exemplifies the use of this API for uploading a File.

```javascript
var object_storage = new ObjectStorageAPI('url of the Object Storage instance');

var fileParts = ['\'<a id="a"\'><b id="b">hey!</\'/b></\'/a>"'];
var myBlob = new Blob(fileParts, { "type" : "text/xml" });

object_storage.uploadFile('folder_name', myBlob, token, {
  file_name: 'myFile.xml',
  onSuccess: function () {
    alert('File uploaded successfully');
  },
  onFailure: function () {
    alert('Something went wrong while uploading the file');
  }
});
```

2.3.1.4.1 KeystoneAPI

A new KeystoneAPI can be instantiated using the following constructor:

KeystoneAPI(url, options)

- url is the url of the Keystone server.
- options:
  - token (String): is the token to used for authenticating request to the Keystone server.
  - use_user_fiware_token (Boolean): make KeystoneAPI to use the token obtained by WireCloud for the current user from the FI-WARE's IdM server. Takes precedence over the token option.

The token and use_user_fiware_token options are optional. When passed to the KeystoneAPI constructor, these values are stored internally and used as the default value in the invocation of its methods. In any case, these options can also be passed to the KeystoneAPI methods.
List all of the tenants in the keystone server available for the authenticated user.

```javascript
getTenants([options])
```

The `onSuccess` callback will receive the list of tenants as the first argument.

**getAuthToken**

Gets an authentication token that permits access to the Object Storage API.

```javascript
getAuthToken([options])
```

Extra options:
- `tenantName` (String): The name of the tenant to be associated to the generated token. Both the `tenantId` and `tenantName` attributes are optional, but should not be specified together.
- `tenantId` (String): The id of the tenant to be associated to the generated token. Both the `tenantId` and `tenantName` attributes are optional, but should not be specified together.

The `onSuccess` callback will receive auth token info as the first argument.

2.3.1.4.2  **ObjectStorageAPI**

A new ObjectStorageAPI can be instantiated using the following constructor:

```javascript
ObjectStorageAPI(url[, options])
```

- `url` is the url of the Object Storage server.
- `options`:
  - `token` (String): is the token to use by default for authenticating requests to the Object Storage server.

All the methods of `ObjectStorageAPI` supports an `options` parameter. This parameter is used, among other things, to pass callbacks functions. This parameter is a JavaScript object containing pairs of key value options. Moreover, all the method of ObjectStorageAPI support at least the following option:
- `token` (String): is the token to used for authenticating the request.

and the following callbacks:
- `onSuccess` is called when the request finishes successfully. The parameters passed to this callback depends on the invoked method.
- `onFailure` is called when the request finish with errors.
- `onComplete` is called when the request finish regardless of whether the request is successful or not.

**createContainer**

Creates a container in which other containers and objects can be stored.

```javascript
createContainer(container[, options])
```

- `container` is the name of the container to create.

**listContainer**

Returns a list of the contents of a container.

```javascript
listContainer(container[, options])
```

- `container` is the name of the container to list.

**getFile**

Retrieves a specified object from the storage system.
getFile(container, file_name[, options])

- **container** is the name of the container where the file is.
- **file_name** is the name of the file to download.

Extra options:
- **response_type** (String, default: "blob"): Valid values are all the supported by the responseType option (see the request options section for more details), except "".

uploadFile
Stores a binary object in the specified location.

uploadFile(container, file[, options])

- **container** is the name of the container where the file is going to be uploaded.
- **file** is the content to be uploaded. Must be an instance of Blob or File.

Extra options:
- **file_name**: name to use for uploading the file. This option is required when passing a Blob as the file argument. This option is not required when passing a File instance as the name is obtained from its name attribute. Anyway, the name passed with this options has precedence over the name attribute of the File instances.

deleteFile
Deletes a specified object from the storage system.

deleteFile(container, file_name[, options])

- **container** is the name of the container where the file is going to be deleted.
- **file_name** is the name of the file to delete.

2.3.1.5  WireCloud IDE

2.3.1.5.1  Installation

- Download the latest version of the "Eclipse IDE for Java EE Developers" package for your platform from the Eclipse's downloads section.
- Download the WireCloud IDE plugin from the FI-WARE PPP Public Files area.
- Install the WireCloud IDE plugin:
  - Help -> Install New Software...
Add a new repository using the Add button

Click Archive and select the WireCloud IDE plugin file downloaded in previous steps
Select the WireCloud IDE entry

Click Next and complete the wizard.

2.3.1.5.2 Adding WireCloud servers

WireCloud IDE support deploying Widget/Operator projects into WireCloud servers. You can manage WireCloud servers from the "Server" view (usually located in the lower half of Eclipse's development screen). If you don't have such a view, you can add it via Window -> Show View -> Servers:
• Right click inside the “Server” tab to open the New Server dialogue:

• In the next step, select WireCloud as the type of server to define (you will find it under the CoNWeTLab category), update the host name of the server where WireCloud is located and give it a name. After filling this info, click Next:
Review the protocol and port configuration. You will also need to provide a client id and a client secret obtained from the IdM (see how to create new OAuth2 applications on the KeyRock’s User and Programmers Guide). You will need to use [WIRECLOUD_SERVER_URL]/oauth2/default_redirect_uri for the callback URL field. Once finished click Next:
The next step is authorising WireCloud IDE to install and uninstall resources on the server. This will require you to log in to the WireCloud server.
And to confirm the authorisation:
After which, a confirmation message should be displayed:
At this step you can choose between clicking Next and selecting a list of projects to upload initially to the server, or clicking Finish as all the required info has been provided. In any case, after finishing the wizard, a new WireCloud server should appear in the “Server” tab:
2.3.1.5.3  How to create Widget/Operator projects from scratch

- Open the new project wizard page:

- And select the Widget/Operator project depending on the type of resource you want to create (you will find them on the Wirecloud Project category):
Give a name to the project and click Finish:
Once created the project, you will obtain the following features in addition to the ones provided directly by Eclipse:

- JavaScript API autocompletion.
- Assistance for editing Widget/Operator descriptions.
- Support for installing and uninstalling them from the configure WireCloud servers.

2.3.1.5.4 Tutorial

- First of all, download this initial code from this link. This code contains a widget example skeleton including basic html/style code.
- Import the downloaded file into a Widget project:
Follow the "Making requests and wiring" tutorial.

Add the widget to the WireCloud server using the Servers view (see the Adding WireCloud servers sections for more info). The first step is opening the "Add and remove ..." view:
• Once opened the "Add and Remove" view, you will need to move the project from the available section to the configured one:

![Add and Remove interface](image)

- Click Finish and the widget will be uploaded to the selected server where you will be able to test your widget following the steps documented on the tutorial.

2.3.2 Additional sources of information

See [The WireCloud website](https://www.wirecloud.eu) for more information. You will find helpful resources such as demo videos, a demo "sandbox" where you will have the opportunity to register and use the tool as a service for free. In that demo deployment of the platform, you will be provided with a catalogue of general purpose widgets, operators and prefab mashups that you can use to experiment and to build new added-value mashups.
3 Light Semantic Composition - User and Programmer Guide

3.1 Introduction
This documentation is related to the Light Semantic Composer whose main target is to facilitate the design process of a Service Composition aided by semantics.

3.1.1 Background and Detail
This User and Programmers Guide relates to the Light Semantic Composition GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

3.2 User Guide
The purpose is to provide the getting-started guide for helping readers to get familiarized with the tools explaining the main functionality and providing some screenshots that will guide the process.

The scope of the document is to provide the functionalities regarding the Lightweight Semantic Composer and therefore how to get a service composition aided by this components. Afterwards, how to deploy and execute it in the Activiti engine. To get more information about the rest of functionalities on the editor and the engine, please visit the Activiti website to enter into detail: http://www.activiti.org/userguide

3.2.1 Login to GE
You can access to the Lightweight Semantic Composition GE that is integrated with the Activiti engine opening the page with Firefox (not Chrome) using the URL "http://<host>:80/ActivitiCompelExplorer2"

It is possible to configure YourBPM to make the user authorization in two different ways:

- Regular login: all users are configured with the BPM-Activiti2 integrated user management option. Compel shows their login page.
- IDM integrated login: All users are validated against the IDM OAuth2.0 system. The role (admin or user) are configured in IDM. The user group is configured with the Compel user management options.

The configuration of the authorization type to be used is done when the application is build. It has been described in the Light Semantic Composition - Installation and Administration Guide how to do it. We copy here and extract from this information.

Configure the following file: $YOUR_BPM/resources/myconfiguration.properties

Configured the following parameter: singlelogon.idmauthentication: true or false. true means that the system will be configured to use the OAuth2.0 servicers from IDM to logon; admin and user roles will be defined in IDM application. false means that the system will use their own login page; admin and user roles will be defined by Activiti application.

3.2.1.1 Regular Login
With a regular login all the user management is being done with the BPM-Activiti2 site. There a 3 predefined users when the software is installed. Log in using the Kermit user to be administrator and be able to create new users and groups.
3.2.1.1.1 Login

With the regular login configuration, when you put the BPM-Activit2 URL ("http://<host>:80/ActivitiCompelExplorer2") in your browser the login page will be presented as shown in

![Login Page](image)

A user existing in the system must be introduced.

3.2.1.1.2 New user and group creation

With the Compel regular login the role of a user (administrator or regular user) with the groups. By default some roles and groups are created when the site is installed, and the administrator and the user roles are initialized during this process. These users are:

- kermit/kermit --> Role: admin
- gonz/gonz --> Role: manager
- fozzie/fozzie --> Role: user

When you edit any group (with the Manage – Groups option) you can see the type of group, it can be security-role or assignment.
All groups of type `security-role` will have different rights when they enter into the Compel site. The administrator and regular belong to this type of group.

All groups of type `assignment` will be able to receive the different programmed notifications.

There are three users to access by default to the integrated environment:

After login, you enter in the application and, depending of the associated role, you can see the allowed functionalities in your Dashboard.

3.2.1.2 **Integrated login**

As we’ve commented in the introduction, the role (admin or user) will be configured in Fi-Ware IDM. The group a user belongs to is configured with the Compel user management option. For each new user we want to grant access we will have to create it in the Fi-Ware IDM site and grant access to the BPM-Activiti2 application. Afterwards, this user will be able to access to the Compel application. The first time the user logs in its profile will be automatically created in the database. From that point on, an administrator will be able to configure the groups the user belongs to. Please, do not try to create the user manually before the first login. We need information retrieved from IDM the first time a user is logging into the system that cannot be created manually.

3.2.1.2.1 **User creation**

To create a new user that has access to Compel application you first must have an account in the Fi-Ware system. Somebody that as permissions to edit the Compel application in Fi-Ware (see next figure) must *Add an Authorized* person.
Four roles can be assigned to an authorized user; see Figure 2: User roles that can be assigned to a user authorized to access to Compel application.

From the four roles the User and Admin make sense within the Compel site. User will be a regular that can receive notification or other tasks depending on the groups he belongs to. As we’ve commented in the Introduction, the user group will be assigned within the application.

Before we can assign any group to the granted user, he will need to login to the system. The first time he logs in, we retrieve the ID assigned to this user in the IDM system and create this user with the same ID in the Compel site.

3.2.1.2.2 Login

When you put the Compel URL [http://compositioned.testbed.fi-ware.eu/compel](http://compositioned.testbed.fi-ware.eu/compel) in your browser you will be automatically redirected to the Fi-Ware login page.
The Fi-Ware login page must inform you that you’re granting access to the Compel application with your credentials. If you have done this step before, and you’ve never logged out from Fi-Ware the browser might skip this page.

Once you’ve signed in, you are requested to authorize Compel to read you public information. This will be done only once.

Once you’ve accepted, you will be redirected to the Compel site. It will depend on your user role assigned in Fi-Ware you will see more or less options in the menu.

3.2.1.2.3  Group assignment

To assign a new group to any user in the Compel site the user must have a role of Admin in Fi-Ware. This kind of user will be able to see the option of Manage – Users. He will be able to select from any user created in the system, see figure below.
In this page the administrator will be able to assign the groups to the selected user. If the needed group don’t exist, we can create it with the Manage – Group option and the Create Group button. Please, make sure you select the assignment type when you create a new group.

3.2.2 Access to the Editor
Clicking on the Process-->Model Workspace, you can access to the Editor.

We can create, import, edit (existing one), copy and delete a model. When we create a “New model”, you have to introduce the name since the diagram will be stored in the database.
When you create a new diagram or edit existing one, you enter in the Lightweight Semantic Composition Editor that is formed by four widgets: Composition Editor, Ontology Browser, Semantic Annotations and Service List. Each of these widgets shows an individual behavior but all of them are related one to each other.

In one screen you can access to all the widgets exposed by the formed description.
3.2.3 Composition Editor

It is the main tool and will drive the composition of services. Dragging and Dropping elements from the palette, the user can design the Service Composition based on the standard notation BPMN2.0. Through this notation, it is possible to interconnect any kind of service that can be aggregated into our composition following the work flow designed by the user. It is important to note that the aim of this tool is at providing the elements to create the flow in the composition.

- We can drag and drop components from the pallet in the left side to the editor panel.
- We can also add properties to each of the components selecting them with the panel in the right side.

For the rest of functionalities, please access to the Activiti documentation.

3.2.4 Discover services through semantic annotation

3.2.4.1 Ontology Browser Widget

Once the user has designed the composition as a set of service tasks[1] connected through a common workflow, the next step is to bind each of these individual tasks with some external service that will actually implement it. This task binding is performed on behalf of the user by a backend semantic matchmaking algorithm which matches composition tasks with real available services. This matching approach requires that the user describes each composition task using light semantic annotations selected from concepts of some domain specific vocabularies or ontologies.

Once the user select a domain specific ontology, the user can define the nature of each of the tasks that participates in the composition, and latter on the application will suggest to the user real services that semantically matches with the description in order to obtain the desired result.
In BPMN a task is a single unit of work that a process consists of. A service task is a task executed by invoking an external service.

- Click on the combo "Select Ontolgy" abd select the ontology
- Click on the button "Load" to see the ontology as a tree

The next step after designing the composite flow and selecting the ontology is to describe each task by attaching to it annotations that refer to concepts taken from the selected ontology. Every task will be defined by annotations such as Functional Requirements, Input, Output and Non Functional Requirements. Once the ontology is loaded and the task is selected in the composition, we can select one concept from the ontology and assign its value to one out of the former four features of the task through the combo box provided by the widget.

- Select one concept on the Ontology
- Scroll down in the widget to the bottom
- Select in the combo "Annotation Type" the desired type of annotation on the selected item
- Click in the button "Add" to add the concept to the annotation.
3.2.4.2 Semantic Annotations Widget

The functionality of this widget is to show the current annotations attached to each task. The user can browse through the task to check how the values of the annotations showed by this widget changes as the user selects another task in the composition.

- Select an item on the design pallet
- Scroll up and down through the widget to see all the Concepts associated to the Annotations on the selected item

On the other hand, once the user has finished annotating a task the button “search” will display in the following widget the services that the tool is able to semantically match with the described task. This action is in real time, since if the user changes one of the annotations and makes again the search the new results will be showed by the Services List Widget.

- Select an item on the design pallet
3.2.5 Service discovery through the Marketplace GE

The user can also discover the services through the integration of the Marketplace GE and the Repository GE. This discovery has a syntactic approach and not a semantic solution as the Semantic Annotations Widget. In this case the repository is outside of the COMPEL and it is not the intention of this section explain how to manage the lifecycle of publishing services (see the documentation for the Marketplace GE (http://forge.fiware.eu/plugins/mediawiki/wiki/fiware/index.php/Marketplace_User_and_Programmer_Guide) and for the Repository GE (http://forge.fiware.eu/plugins/mediawiki/wiki/fiware/index.php/Repository_User_and_Programmer_Guide)). Hence, we assume that the services has been created in the Repository GE and they have been published in the Marketplace GE. The description of the repository has to be described through the Unified Service Description Language (USDL) (http://linked-usdl.org/).

The user can select the Marketplace Link into the Ontology Browser Widget.

The pop-up appears with the search form.
The user can close the pop-up clicking the close button. The user has to introduce a keyword in order to search in the Marketplace GE:

- if the user don't introduce a keyword, the system shows an error.

- if the user introduce a keyword, the system call the remote Marketplace in order to execute the search, and it shows the results:
  - If there are not services, the system shows a message.
- if there are services, the system shows the list of services which the Marketplace has found.

The user can see the list of the service. Every service has the name, the associated store and the URL, in order to see more details the user has to click in the "More Info" button.
We can consider three main parts.

- At the top, the user can search again or close the pop-up.
- In the middle, the user can see the details of the services (logo, short description, the description, the associated page and the USDL description link). The user can come back, clicking the “Back” button.
- At the bottom, the user can see the link of the WSDL and the list of the associated methods.

The user has to select the method that should be bound with the Activiti task and click in the button “Bind”
The system indicates that this method has been bound to the selected task.

And the user can see that this task is bound with the method in the Service Operation Widget.
In case it doesn't work, please make sure you're able to access to http://appsnserv.testbed.fi-ware.eu/FiwareMarketplace/v1/search/offerings/fulltext/logistic, where 'logistic' is the keyword that you want to search in the marketplace (for more information about the search, you can see this section). It is the marketplace URL, and the Light Sematic Composition needs to have access to it.

3.2.6 Services List Widget

This widget has two main functionalities. The first one is to show the list of services that semantically matches with the annotations made by the user in the selected task or the service that has been selected in the Marketplace. The name of the service, the operation and the link to the description of the service will be shown in a table mode. The second functionality is related to the selection of one the services that comes from the services list. Only one service will be executed by the task, so it is the user who decides with a selective combo which of the candidate services will be bound to the task.

- Select an item on the design pallet
- Make all the annotations on the item following the above instructions
- Select one of the listed services and click on "Bind" to associate the item to a concrete service
3.2.7  Save the process

When you have finished the design of your process, you can save it, clicking on the icon \( \text{Save} \). You have to confirm or change the data.

Now, you can close the Editor, clicking on the icon \( \text{Close} \). If you try to close the editor without to save the changes, the Editor detects it and asks you to confirm.
The process has been stored in the database, so you can modify it again, maintaining all the annotation and binding (remember to click on the annotated task).

### 3.2.8 Generate BPMN 2.0 file

You can also generate a BPMN2.0 file compliant with the associated services that you have bound in the Editor. This functionality introduces all the standard tags in the generated file, binding the WS indicated in the process (through the associated WSDL).

Go to the Model Action --> Export Model Compel.
The application generates the executable file with all the binding task description.

You can download or open it. In the file, the application has introduced the references of the Web Services.
<import namespace="http://atosresearch.eu/
importType="http://schemas.xmlsoap.org/wsdl/">

<process isExecutable="true" name="TestWS_Stock" id="TestWS_Stock">
  <serviceTask implementation="##WebService"
operationRef="tns:isAvailableOperation" name="Stock WS" id="sid-B4AFEC AF-A750-4723-BDB5-40FE2ED3D8E2">
    <ioSpecification>
      <dataInput itemSubjectRef="tns:isAvailableItem"
id="isAvailableInput"/>
      <dataOutput itemSubjectRef="tns:isAvailableResponseItem"
 id="isAvailableResponseOutput"/>
      <inputSet>
        <dataInputRefs>isAvailableInput</dataInputRefs>
      </inputSet>
      <outputSet>
        <dataOutputRefs>isAvailableResponseOutput</dataOutputRefs>
      </outputSet>
      <dataInputAssociation>
        <sourceRef>dataInputOfidReference</sourceRef>
        <targetRef>idReference</targetRef>
      </dataInputAssociation>
      <dataOutputAssociation>
        <sourceRef>availabilityResult</sourceRef>
        <targetRef>dataOutputOfisAvailableOperation</targetRef>
      </dataOutputAssociation>
    </ioSpecification>
</serviceTask>
</process>

<itemDefinition structureRef="seiplab:isAvailable"
 id="isAvailableItem"/>

<itemDefinition structureRef="seiplab:isAvailableResponse"
 id="isAvailableResponseItem"/>

<message itemRef="tns:isAvailableItem" id="isAvailableMessage"/>

<message itemRef="tns:isAvailableResponseItem"
 id="isAvailableResponseMessage"/>

<interface name="StockServiceInterface"
implementationRef="seiplab:StockService" id="StockServiceInterface">
  <operation name="isAvailableOperation"
implementationRef="seiplab:isAvailable" id="isAvailableOperation">
3.2.9 Deploy and execute the BPMN 2.0 file

We are going to detail a simple example to deploy and execute a process with a ServiceTask. It is mandatory that the called services (WS) will be available and ready to be executed.

3.2.9.1 Build the BPMN process

- Create a "New model" and enter the name "TestWS".
- Design a BPMN process: StartEvent, ServiceTask and EndEvent.
- Go to the "Light-weighted Semantic Composition" panel situated at right of the window. In the Top Widget, select the logistic ontology (Logistic-v2) and load it.
- The next step is to select a ServiceTask and make semantic annotations to the service task.
- Once the task has been selected, select the concept you want to associate with the task in the
ontology browser.

- Scroll down in the widget to the top where you can select the Annotation Type you want to make to the Task (input, Output, Functional Requirement and Non F Req).
- Select the following triplet to annotate your ServiceTask:

  | Functionality: FunctionalClassificationRoot -- ActivityType -- Storage |
  | Input: Entity -- SpatialEntity -- MoveableResource |
  | Output: Entity -- SpatialEntity -- isAvailableIn |

- Click on the "Add" button for every Annotation Type.

- Click the "Search" button in the middle widget.
- Select the service named Stock (wsdl) and bind it.

- Select the Start Event properties to introduce the Activity variables. Click in the Properties (Start Event) and select the "Form Properties". Introduce the variable reference (type String).
- Save the model and comeback to the "Model Work Space"
- Go to the Model Action --> Export Model Compel. The application generates the executable file with all the binding task description.
- All the associated structure to the WSDL has been introduced, however it is necessary to map the Activity variable with the inputs of the ServiceTask.
  - Associate the input of the ServiceTask to the name of the Activiti input variable (form associated to the StartEvent), you have to change:

  ```xml
  variable="Reference" --&gt; variable="dataInputOfidReference"
  ```

```
<serviceTask implementation="##WebService" operationRef="tns:isAvailableOperation" name="Stock WS" id="sid-B4AFECAF-A750-4723-BDB5-40FE2ED3D8E2">
  <ioSpecification>
    <dataInput itemSubjectRef="tns:isAvailableItem" id="isAvailableInput"/>
    <dataOutput itemSubjectRef="tns:isAvailableResponseItem" id="isAvailableResponseOutput"/>
    <inputSet>
      <dataInputRefs>isAvailableInput</dataInputRefs>
    </inputSet>
    <outputSet>
      <dataOutputRefs>isAvailableResponseOutput</dataOutputRefs>
    </outputSet>
  </ioSpecification>
  <dataInputAssociation>
    <sourceRef>dataInputOfidReference</sourceRef>
    <targetRef>idReference</targetRef>
  </dataInputAssociation>
  <dataOutputAssociation>
    <sourceRef>availabilityResult</sourceRef>
  </dataOutputAssociation>
</serviceTask>
```
<targetRef dataOutputOfisAvailableOperation</targetRef>
</dataOutputAssociation>

<startEvent id="sid-FB6B7AF7-13F2-4085-B0C1-0F4F9EC686BD">
<extensionElements>
    <activiti:formProperty id="1" name="Reference" type="string" variable="dataInputOfidReference"/>
</extensionElements>
</startEvent>

- Save the file bpmn20 in your local disk.

3.2.9.2 **Deploy the BPMN process**

Afterward to map the variables, the file is ready to be executed, so we are going to deploy it.

- Select "Manage"--->"Deployment"--->"Upload new"

- Appear a popup window and select the new file BPMN20

- The system deploys and uploads the new process. The following screen appears:
You can go to the deployed process:

- Click in the link TestWS_Stock
- Select "Process"->"Deployed process definitions" and choose the new process TestWS_Stock.

3.2.9.3 **Execute the BPMN process**
- Select "Start process", the system asks for the variable Reference.

- Introduce the variable value and submit.
- The system indicates that the process has been started correctly.
The following logs appear in the console of the WS and the process finishes:

```plaintext
idReference request: <ReferenceValue>
isAvailable: <if the ReferenceValue=0001 then isAvailable=1 else isAvailable=0>
```

### 3.2.10 Limitations and recommendations

Currently the Semantic Web Service (SWS) descriptions, to annotate the services, are completely manual, so, it is necessary to use directly the posm files in order to introduce annotations in the new services. In next releases, it is planned to simplify it and create a graphical user interface, in order to introduce these annotations in the description of the services using different vocabularies.

In the current version, the data flow mappings have also to be introduced manually in the generated BPMN file and, in next releases, it is planned to simplify and improve them.

Finally, the development roadmap for this GE implementation considers important changes in the Light Semantic Composition Editor UI, in particular with regards to the semantic widgets and the integration with other GE implementations (Marketplace, Repository), whereby new versions of this document will detail those changes.

### 3.3 Programmer guide

Since the Light Semantic Composition is a Generic Enabler which provides mainly (UI) frontend functionality to different roles (e.g. business analysts and services integrators), we do not distinguish between the User and Programmers Guide. Please refer to the User Guide section for more information.
4 Marketplace - User and Programmer Guide

4.1 Introduction

This document describes the necessary steps to develop a software application or a user interface which makes use of the marketplace backend functionality. The Marketplace API is based on REST principles and generally returns XML or JSON encoded responses. Since REST is independent of a concrete programming language, you just have to know how to make an HTTP request in the programming language of your choice. You must authenticated all your requests to the Marketplace API with username and password.

4.1.1 Background and Detail

In general, this page relates to the Marketplace GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

This page contains Users and Programmers information for the reference implementation of the following Marketplace Open API Specifications:

- Marketplace Offerings Open RESTful API Specification
- Marketplace Registration Open RESTful API Specification
- Marketplace Search Open RESTful API Specification
- Marketplace Review and Rating Open RESTful API Specification
- Marketplace Recommendation Open RESTful API Specification

4.2 User Guide

In general we have to distinguish between the backend components of the Marketplace Generic Enabler, which just provide APIs to other applications, and the end-user-facing Pricing Component. For the pure backend components, we do not distinguish between the User and Programmers guide. For the end-user-facing Pricing Component, which is also part of the Marketplace GE, a User Guide is provided here.

To get more information about the core backend components, please refer to the Programmers Guide section, for getting more information about the Pricing Support Component continue reading with the following chapter.

4.2.1 Using the Pricing Support Component

4.2.1.1 Introduction

The FI-WARE Pricing Support/ Simulator supports the decision-making processes involved in the pricing choices a software provider (from now on simply called “the vendor”) must make for its offerings. Some characteristics of the pricing decision process – its complexity and its being fundamentally the outcome of a series of economic interactions between autonomous market agents – make it an ideal candidate to be investigated by means of a simulation approach called Agent-Based Modeling and Simulation. Using the Pricing Support Component, simulation experiments based on the Agent-Based Modeling and Simulation approach can be conducted to support important business decisions, such as selecting an optimal price level or determining a balanced price / product configuration combination (i.e., offering the right value, in terms of attributes and characteristics of the offered software product or service, for the money customers must spend in acquiring it).
Roughly speaking, a pricing decision must concurrently take into account three aspects: the customers’ preferences, the competitors’ own pricing decisions, and the vendor’s own costs in providing the software solution. The Pricing Support underlying market models parameterize to at least some extent all of them. Relying on an intuitive Web-based interface, a user has thus full control of the experimental setting in which to investigate pricing decisions and can explore how different price/product configurations would lead to different outcomes in terms of unit sales, revenues, and profits.

4.2.1.2 Overview of the User Interface

The three main components of the Pricing Support GUI are, from top to bottom, the menu bar, the tabs, and the main panel. The menu bar allows to quickly access functionalities related to importing and loading pre-defined market scenarios, setting preferences, and viewing the application help. The tabs are used to navigate the panes which allow to fully configure and run a simulation experiment. Different panes are linked to the different tabs and are displayed in the main panel, where appropriate GUI elements enable the user to insert required data, configure parameters, and inspect the simulation results.

![Pricing Simulator - Home Screen](image)

A generic workflow for a user will encompass defining the attribute space first (the way software product and services can be described and distinguished from each other in the market), subsequently configuring the market agents (competitors and customers), and then finally running the experiments to explore the outcomes of alternative pricing decisions.

4.2.1.3 Attribute Space Pane

In the model used by the Pricing Support Component, a software product or service is described as a set of attributes. Each can take one of a series of attribute levels. An attribute can be looked at as an ideally configurable characteristic of a product or service (e.g., which mobile platforms are supported), and the attribute levels as values which can be selected for a concrete instance or implementation of an offering (e.g., iOS, Android, or both). In the Attribute Space pane, accessible by clicking on the Attribute Space tab, the user can easily add, modify, or remove attributes and attribute levels using the provided input fields and buttons.
Pricing Simulator - Attribute Space

4.2.1.4 Market Scenario Pane

In the Market Scenario pane, accessible by clicking on the Market Scenario tab, the user defines the market environment in which a price decision is to be tested. The market environment consists of the economic agents populating the two sides of the market and the entities exchanged by those agents in their economic transactions. The agents are respectively the vendor and its competitors on the supply side, and current or potential customers on the demand side. The entities are the product/service offerings present in the market during the simulation, where an offering is just a product/service (defined through a specific combination of attribute levels as explained above) with an attached price.

To avoid overwhelming the users with too many parameters and corresponding input fields, the two ideal sides of the market are described in two different sub-panes accessible with the two second-level tabs Supply and Demand.

4.2.1.4.1 Supply

Here the user can configure own offerings and specify other providers present in the market to describe competitors. Offerings are always configured in terms of the previously defined attributes and attribute levels (please note the user can go back to the Attribute Space pane at any time and change them, if needed). Own offerings can additionally be characterized by some cost variables to describe the expenses incurred by the vendor in providing the offering in the market. Specifying these cost parameters is necessary if the user wants to investigate the effects of pricing decisions on the generated profits (which will otherwise be equal to the generated revenues). Competitive offerings require specifying a price as well.
4.2.1.4.2 Demand

Here the user can define the market segmentation in terms of customer groups and their respective preference structures. A group of customer with a common preference structure is called a Segment. For each segment an identification name and the size (the number of customers it includes) can be specified. The fundamental characteristic of a customer segment is, however, its preference structure (sometimes called value system in the marketing field). A segment’s preference structure can be easily configured by clicking on the edit button, which will open a pop-up window with the appropriate configuration fields.

Once again, these are based on the previously defined attributes and attribute levels, given that a full preference structure comprises the preferences of customers towards both attributes (i.e., the relative importance of the product/service characteristics) and attribute values (i.e., the predilection for one concrete instance or implementation of a certain attribute). To specify how different attributes rate against each other in a customer’s value system, attribute weights can be given for each attribute. To specify how different attribute levels rate against each other, the Scores fields must be used.
The values the user is supposed to employ may be the user's own estimations or come from a customer survey, a structured research study (such as a conjoint analysis study), or experts' assessments.

4.2.1.5 **Experiment Pane**

Once the user has defined the attribute space and the market scenario, the Experiment pane is where generic simulation parameters and the pricing “research question” are configured, and the experiment’s results visualized. The user is free to choose the length of the simulation (field “Execution length”) and the number of stochastic replications to be conducted in the experiment (field “Number of runs”). Subsequently, the user must decide what kind of experiment to perform, in other words, how to formalize the pricing questions.

Two are the possibilities currently offered by the Pricing Support Component. Selecting “static pricing analysis” the user can define an interval of prices to be evaluated. The (inclusive) interval bounds can be specified moving the two knobs on the “Price” slider, which will also display as a term of reference the competitive offerings’ prices (if any offering was specified in the Market Scenario Pane). The granularity of the analysis will be determined by the “Price step” field. For example, experimenting with an interval from 0€ to 10€ with step of 3€, the prices 0€, 3€, 6€, 9€ will be used for individual simulation batches (a batch being a set of simulation runs of size given by the “Number of runs” parameter).

The second possibility is to simulate dynamic strategies (dynamic pricing analysis), that is, to simulate different price trajectories over time. The vendor, for example, could be interested in the outcomes of the dual strategies of penetration pricing (low price that increases over time) and skimming (high price that decreases over time). First of all, the alternative pricing strategies need to be defined. This is easily accomplished by first adding an “empty” strategy (using the button “Add strategy” and then defining the individual price moves composing the strategy in the pop-up window accessible via the button “Edit strategy”. Once defined, alternative pricing strategies will be simulated and compared just like the different price levels in the interval of a static pricing analysis.
At this point the user can run the experiment clicking on the run button and, once all the simulation replications are executed, inspect the results of different pricing decisions. Simulation results are shown both graphically and in tabular form in a pop-up window.

4.2.1.6 Additional Functionalities

It is possible to load predefined scenario files prepared with Microsoft Excel. These scenarios can then be further modified as desired, according to the above-mentioned guidelines.

4.3 Programmer Guide

To give you a feeling of how the Marketplace works and how you can interact with the system let us take a look at some examples, realized with the command line tool cURL and in Java. 'cURL' is a command which can be used to perform any kind of HTTP operation - and therefore is also usable for the Marketplace. The library libcurl enables the integration in C programs as well.
4.3.1 Accessing the Marketplace with cURL

In the following, a couple of ways are described how to access the Marketplace with the command line tool cURL.

- `[SERVER_URL]` is the URL on which the Marketplace is deployed (e.g. localhost:8080)
- `[STORE_NAME]` is the name of your store registered at the Marketplace.

4.3.1.1 Create an offering with cURL

This example shows how to create an offering with the command line tool 'cURL':

- Create a message body and save it to a file named `messageBody.xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource name="myService">
    <url>[URL_TO_RESOURCE]</url>
</resource>
```

- send the request to the server

```
curl -v -H "Content-Type: application/xml" -X PUT --data 
"@messageBody.xml" -u "demo:demo"
http://[SERVER_URL]/FiwareMarketplace/v1/offering/store/[STORE_NAME]/offering
```

- You should obtain the following message

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Server auth using Basic with user 'demo'
> PUT /FiwareMarketplace/v1/offering/store/testStoreAB/offering HTTP/1.1
> Authorization: Basic ZGVtbzpkZW1v
> User-Agent: curl/7.23.1 (i386-pc-win32) libcurl/7.23.1 zlib/1.2.5
> Host: [SERVER_URL]
> Accept: */*
> Content-Type: application/xml
> Content-Length: 202

* upload completely sent off: 202 out of 202 bytes
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Set-Cookie: JSESSIONID=0677F2F6E8CC6276D3CE59382CDBF887;
Path=/FiwareMarketplace
< Content-Length: 0
< Date: Wed, 11 Jul 2012 09:49:27 GMT
```
In this case the HTTP PUT operation is used. The Marketplace uses the Create Retrieve Update Delete (CRUD) operations which map almost to the HTTP verbs PUT, GET, POST and DELETE.

4.3.1.2 Full Text Search cURL

This examples demonstrates how to perform a full text search for offerings.

Send a search request to the server:

```
curl -v -H "Content-Type: application/xml" -X GET -u "demo:demo" http://[SERVER_URL]/FiwareMarketplace/v1/search/offerings/fulltext/test
```

- You should obtain a result body similar to the result shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><searchresults><searchresult><matches><match><literal>Test Service</literal><luceneScore>1.0</luceneScore><text>Test Text</text></match></matches><service name="TestService">
...
</searchresult></searchresults>
```

4.3.2 Accessing the Marketplace with Java

4.3.2.1 Create Client Request with username and password

- You need a client request object to perform operations on the marketplace. For that you have to authenticate yourself against the Marketplace using username and password:

```java
String user = "USERNAME";
String password = "PASSWORD";
String endpoint = "SERVER_URL"

URI uri=null;
```
try {
    uri = new URI(uriString);
} catch (URISyntaxException e1) {
    e1.printStackTrace();
}

Credentials credentials = new UsernamePasswordCredentials(user, pwd);
HttpClient httpClient = new HttpClient();
httpClient.getState().setCredentials(AuthScope.ANY, credentials);
httpClient.getParams().setAuthenticationPreemptive(true);

ClientExecutor clientExecutor = new ApacheHttpClientExecutor(httpClient);

ClientRequestFactory fac = new ClientRequestFactory(clientExecutor, uri);
ClientRequest request = fac.createRequest(endpoint+"/offering/store/"+storeName+"/offering");

- Now you can perform different operations on the defined endpoint. In this example the HTTP PUT operation is used.

4.3.2.2 Create a Resource

request.body("application/xml", input);
ClientResponse<String> response;
response = request.put(String.class);
if(response.getStatus() == 201){
    return true;
}

- According to the REST design principles, you can also use request.post(...), request.get(), request.delete() on an endpoint to update, get or delete resources.

4.3.3 Using the Review and Rating Component of the Marketplace

The Review and Rating component allows users of the marketplace to give textual and star-rating feedback for user defined objects. These objects are not necessarily, but may be also a part of the marketplace itself (e.g. stores or services). Reviews of users and their ratings are being used to improve the quality of the recommendation component.

In the following we assume that you have the command line tool 'cURL' installed on your system:
4.3.3.1 **Creating Reviews and Ratings**

4.3.3.1.1 **Create a new Rating Object Category:**

At first we have to create a new Object Category. Object Categories are mainly used to cluster user review and ratings.

```sh
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]
```

- **Example:** If you have a marketplace for web services you would create different object categories for different service types (Weather Services, Location Services, Traffic Services)

```sh
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices
```

4.3.3.1.2 **Create a new Rating Category:**

Rating Categories are tight to Object Categories and are used to differentiate ratings.

```sh
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/category/[RatingCategoryName]
```

- **Example:** You want to enable the users of your Service Marketplace to rate "Weather Services" along different categories (accuracy, price, documentation)

```sh
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/category/accuracy
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/category/price
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/category/documentation
```

4.3.3.1.3 **Create a new Rating Object:**

Rating Objects are concrete objects which can be rated. A Rating Object is tight to exactly one Object Category.

```sh
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/[ObjectName]
```

- **Example:** Create a concrete Weather Service Object (BestWeather):

```sh
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/BestWeather
```

4.3.3.1.4 **Create a new Rating:**

Now we have to rate the created object along the defined rating categories. In the first step, we have to ask the system to create the rating and give us an ID.
You will receive a rating ID in the result.

- **Example**: Create the Rating and receive the ID:

```plaintext
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]/rating/
```

4.3.3.1.5  **Create Rating for Category**:

Use the ID obtained in the "Create a new Rating" step to rate an object.

```plaintext
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]/rating/[RatingId]/category/[RatingCategoryName]/stars/[STARS]
```

- **Example**: Now we can use the obtained ID to rate the "BestWeather" Service along our three categories.

```plaintext
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/BestWeather/rating/[ID_OBTAINED]/category/accuracy/5
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/BestWeather/rating/[ID_OBTAINED]/category/price/4
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/BestWeather/rating/[ID_OBTAINED]/category/documentation/4
```

4.3.3.1.6  **Create Textual Review**:

To Create a textual review for a rating just use the Rating ID and post/put your review message to the textual review endpoint.

```plaintext
curl.exe url -v -H "Content-Type: application/xml" -X PUT -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]/rating/[RatingId]/textualReview/[Review_Message]
```

- **Example**: Now we can use the obtained ID to rate the "BestWeather" Service along our three categories.

```plaintext
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/WeatherServices/object/BestWeather/rating/[ID_OBTAINED]/textualReview/Very Good Service
```

4.3.3.2  **Receiving Reviews and Ratings**

For Receiving Review and Ratings as well as all other objects responsible for the Review and Rating system, use one of the following commands:
4.3.3.2.1  Get a Rating:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]/rating/[RatingId]

4.3.3.2.2  Get a Category:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/category/[RatingCategoryName]/Quality

4.3.3.2.3  Get a new Rating Object:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]

4.3.3.2.4  Get a new Rating Object Category:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]

4.3.3.2.5  All Ratings for an Object:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/object/[ObjectName]/ratings

4.3.3.2.6  All Objects for an Object Category:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/objects

4.3.3.2.7  All Categories for an Object Category:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]/categories

4.3.3.2.8  All Available Object Categories:

curl.exe url -v -H "Content-Type: application/xml" -X GET -u
"username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategories
4.3.3.3 **Deleting Reviews and Ratings**

For deleting reviews, ratings, categories, object categories and rating objects choose one of the following commands. Please keep in mind that all deletion operations are cascading - use them with care.

4.3.3.3.1 **Delete a Rating:**

```bash
curl.exe url -v -H "Content-Type: application/xml" -X DELETE -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectName]/object/[ObjectName]/rating/[RatingId]
```

4.3.3.3.2 **Delete a Category:**

```bash
curl.exe url -v -H "Content-Type: application/xml" -X DELETE -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectName]/ratingCategory/[ObjectCategoryName]/Quality
```

4.3.3.3.3 **Delete a new Rating Object:**

```bash
curl.exe url -v -H "Content-Type: application/xml" -X DELETE -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectName]/[ObjectName]
```

4.3.3.3.4 **Delete a new Rating Object Category:**

```bash
curl.exe url -v -H "Content-Type: application/xml" -X DELETE -u "username:password"
http://[SERVER:PORT]/FiwareMarketplace/v1/rating/objectCategory/[ObjectCategoryName]
```

4.3.4 Using the Rating based Recommendation Component of the Marketplace

This component provides the user service recommendations based on ratings.

4.3.4.1.1 **Get available Object Categories:**

```bash
http://[SERVER:PORT]/FiwareMarketplace/v1/recommendation/objectCategories
```

4.3.4.1.2 **Get Recommendations for an Object Category:**

```bash
http://[SERVER:PORT]/FiwareMarketplace/v1/recommendation/objectCategory/[ObjectCategoryName]/
```
5 Mediator - User and Programmer Guide

5.1 Introduction
The Mediator is based on the open source packages WSO2 ESB and Apache Camel. Moreover, it includes custom TI code and custom virtual proxy configuration specifically developed for FIWARE. The current version of the Mediator does not provide remote API so the creation of mediation services can be only performed using WSO2 ESB functionalities of the Mediator and its custom pages for coding Apache Routes through Java classes, detailed below (or any other open source tools focused on coding Apache Camel routes through Java classes). For these reasons, the relevant User and Programming Guide, for the current release of the Mediator, are those related to the open source packages which the Mediator is based on.

1. Apache Synapse Virtual proxy
2. Apache Camel routes

5.1.1.1 Background and Detail
This User and Programmers Guide relates to the Mediator GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

5.2 User Guide
The user and programmer guide of the two open source technologies mentioned in the Introduction can be found respectively at:

wso2-esb 4.0.0. User Guide
http://wso2.org/project/esb/java/4.0.0/docs/user_guide.html

Apache-camel 2.7.0 Manual

For further aspect of this component, please refer to the section "Programmer Guide".

5.3 Programmer Guide
5.3.1 Camel Routes Handling
Mediator Web GUI has pages dedicated to Apache Camel Routes coding and handling. These pages allows to

- view currently deployed Mediator Apache Routes
- view an Apache Route code in Java and in XML
- edit an Apache Route Java code (selecting "Full Java" and then "edit")
- add an Apache Route to the Mediator, uploading an archive file. The archive (e.g.: routeName.zip) must structured as:
  - jar libraries in the root folder - all the libraries used by the Apache Route must be here;
  - a directory (the name is the route name) in the root folder;
  - source java files and resources, inside the preceding folder, with package structure.
  - example image:
5.3.2 User Logs

In the Monitoring tab, the User Logs custom entry shows a page with additional log informations:

- Requests per User
- Requests per User/Service
- Last Requests

To log this information for a WSO2 proxy service you have to add the custom LoggingMediator at the start of the inSequence of the proxy service. Usage example:

```xml
<inSequence>
  <class name="it.telecomitalia.ictlab.logging.mediator.LoggingMediator">
    <property name="otherInfoXpath" />
    <property name="dbConnection">
      <url>jdbc:h2:repository/database/WSO2CARBON_DB</url>
      <username>wso2carbon</username>
      <password>wso2carbon</password>
      <driver>org.h2.Driver</driver>
    </property>
  </class>
  [...]
</inSequence>
```

5.3.3 Examples

In the following we describe examples of the Mediation Services definition via WSO2 ESB and Apache Camel, written following the above guides.

5.3.3.1 WSO2 ESB

Example 1: Protocol Transformation TCP2HTTP
Example 2: WS-Security
Virtual proxy configuration that adds WS-Security to the unsecured target service "ServiceExample"

```xml
<proxy xmlns="http://ws.apache.org/ns/synapse" name="SecuredServiceExampleProxy" transports="https" startOnLoad="true">
    <target>
        <inSequence>
            <header xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsssecurity-secext-1.0.xsd" name="wsse:Security" action="remove" />
        </inSequence>
        <endpoint>
            <address uri="http://<ServiceExample URL>" />
        </endpoint>
        <outSequence>
            <send />
        </outSequence>
    </target>
    <enableSec />
    <policy key="conf:/repository/axis2/service-groups/SecuredServiceExampleProxy/services/SecuredServiceExampleProxy/policies/UTOverTransport" />
</proxy>
```

Example 2.1
The above example separating endpoint and proxy definition and adding validation, log, wsdl published handling.

Endpoint definition:

```xml
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="posAdapter">
    <address uri="http://polaris.csel.it:9763/services/PosAdapter" >
        <suspendOnFailure>
```
Proxy Service definition:

```xml
<proxy xmlns="http://ws.apache.org/ns/synapse" name="PosAdapterProxy" transports="https" statistics="enable" trace="enable" startOnLoad="true">
  <target endpoint="posAdapter">
    <inSequence>
      <class name="it.telecomitalia.ictlab.logging.mediator.LoggingMediator">
        <property name="otherInfoXpath">
          <otherInfoXpath />
        </property>
        <property name="dbConnection">
          <dbConnection>
            <url>jdbc:h2:repository/database/WSO2CARBON_DB</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>
            <driver>org.h2.Driver</driver>
          </dbConnection>
        </property>
      </class>
    </inSequence>
    <log level="full" />
    <validate>
```

```xml
  </validate>
</proxy>
```
<schema key="gov:/schemas/it/telecomitalia/ictlab/posadapterschema/PosAdapterSchema.xsd" />

<resource location="/.../ictcommondatamodelschema/ICTCommonDataModelSchema.xsd"
key="gov:/schemas/it/telecomitalia/ictlab/ictcommondatamodelschema/ICTCommonDataModelSchema.xsd" />

<on-fail>
  <sequence
  key="conf:/repository/synapse/default/sequences/validationFaultSequence"
  />
</on-fail>
</validate>

<header xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" name="wsse:Security" action="remove" />
</inSequence>

<outSequence>
  <log level="full" />
  <validate>
    <schema key="gov:/schemas/it/telecomitalia/ictlab/posadapterschema/PosAdapterSchema.xsd" />
    <resource location="/.../ictcommondatamodelschema/ICTCommonDataModelSchema.xsd"
key="gov:/schemas/it/telecomitalia/ictlab/ictcommondatamodelschema/ICTCommonDataModelSchema.xsd" />
    <on-fail>
      <sequence
      key="conf:/repository/synapse/default/sequences/validationLogSequence"
      />
    </on-fail>
  </validate>
  <send />
</outSequence>
</target>

<publishWSDL
key="gov:/wsdls/it/telecomitalia/ictlab/posadapter/_2009_11/posAdapter.wsdl">
  <resource
  location="/.../.../.../.../schemas/it/telecomitalia/ictlab/ictcommondatamodelschema/ICTCommonDataModelSchema.xsd"
key="gov:/schemas/it/telecomitalia/ictlab/ictcommondatamodelschema/ICTCommonDataModelSchema.xsd" />
  <resource
  location="/.../.../.../.../schemas/it/telecomitalia/ictlab/posadapterschema/PosAdapterSchema.xsd"

D.3.4.3: FI-WARE User and Programmer Guide 118
Example 3: Custom Java Mediator

In order to use custom mediation task inside a mediation service you have to use the custom built-in mediator in the proxy service configuration, for example:

```xml
<class
name="it.telecomitalia.fiware.mediator.irrigation.IrrigationMediationTask"
/>
```

The java class shall extends the abstract class: org.apache.synapse.mediators.AbstractMediator and override the method public boolean mediate(MessageContext context)

To get an example of custom mediator see the webinar folder into the fiware svn repository: https://forge.fi-ware.eu/scmrepos/svn/apps/trunk/mediator-usecase/webinar-usecase-201211

Example 4: data transformations

- To use a xslt transformation
- To use xquery language
- To use the xpath built in capability in order to manipulate the message
- To use a custom java mediator - see Example 3

Mediation Service Example

```xml
<proxy xmlns="http://ws.apache.org/ns/synapse"
name="company4WeatherMonitorAndReact" transports="http"
statistics="disable" trace="disable" startOnLoad="true">
	<target endpoint="mailAdapter">
		<inSequence>
			<log level="full" />
			<class
name="it.telecomitalia.fiware.mediator.irrigation.IrrigationMediationTask"
/>
			<log level="full" />
			<xslt
key="gov:/transformations/fiwareCreateMailForWeatherEvent.xslt">
			<property name="mailRecipient" value="[target_email_address]"
				/>
			</xslt>
		</inSequence>
	</target>
</proxy>
```
In this case the service mediator performs a xslt transformation, apply a custom mediation task written in java code and send the transformed message to a mail service.

**Example 4.1: XSLT transformation**

The XSLT mediation capability applies the specified XSLT transformation to the selected element of the current message payload. The source attribute specifies which element to be selected to apply the given XSLT transformation. In the case where the source element is not specified, it uses the first child of the soap body as the selected element. Optionally parameters could be passed into the transformations through the 'property' elements. These properties are corresponding to the XSL parameters and can be accessed during transformation by `<xsl:param name="[the name of the property]"/>`.

Finally, the 'resource' element can be used to resolve XSLT imports and includes from the repository. You can upload XSLT files on the Registry with the "Add Resource" functionality of the Registry. Syntax

```xml
<xslt key="string" [source="xpath"]>
    <property name="string" (value="literal" | expression="xpath")/>
    <resource location="string" key="string"/>
</xslt>
```

- **Key**: The registry key to refer the xslt. This supports static and dynamic keys.
- **Source**: Specify in which part of message (specified in xpath) the xslt should be applied. Default is the SOAP body of the message.
- **Properties of the XSLT mediator**: Manage the properties which would be referred from the xslt in transformation (using get-property(prop-name) xpath extension function).
  - Property Name: Name of the property.
  - Property Type: Whether it is a static value or an xpath expression.
  - Value/ Expression: The static value or the xpath expression.
  - NSEditor: Specify the namespaces that are used in the xpath expression.

**Example 4.2: XQuery mediation capability**

The XQuery mediation capability can be used to perform an XQuery transformation. The 'key' attribute specifies the registry key for looking up XQuery script that going to be used for define transformation. You can upload XSLT files on the Registry with the "Add Resource" functionality of the Registry. The optional 'target' attribute specifies the node of the SOAP message that should be transformed. In case where the target value is not specified, then the first child of the SOAP body is selected. The 'variable' elements define a variable that could be bound to the dynamic context of the XQuery engine in order to access those variables during the XQuery script invocation.

In the variable definition, it is possible to specify just a literal value, or an XPath expression over the payload, or even specify a registry key or a registry key combined with an XPath expression that selects the value of the variable. Using key attribute of the variable, it is possible to bind an
XML document located in the registry so that in the transformation that too can be used. The name of the variable corresponds to the name of variable declaration in the XQuery script. The type of the variable must be a valid type defined by the JSR-000225 (XQJ API).

**Supported Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XQItemType.XQBASETYPE_INT -&gt; INT</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_INTEGER -&gt; INTEGER</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_BOOLEAN -&gt; BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_BYTE -&gt; BYTE</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_DOUBLE -&gt; DOUBLE</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_SHORT -&gt; SHORT</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_LONG -&gt; LONG</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_FLOAT -&gt; FLOAT</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQBASETYPE_STRING -&gt; STRING</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQITEMKIND_DOCUMENT -&gt; DOCUMENT</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQITEMKIND_DOCUMENT_ELEMENT -&gt; DOCUMENT_ELEMENT</td>
<td></td>
</tr>
<tr>
<td>XQItemType.XQITEMKIND_ELEMENT -&gt; ELEMENT</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```xml
<xquery key="string" [target="xpath"]>
  <variable name="string" type="string" [key="string"] [expression="xpath"] [value="string"]//>
</xquery>
```

- **Key**: The key that represent the xquery transformation.
- **Target**: (optional) Specify the node of the message that should be transformed using an xpath expression. (default to the first child of the SOAP body) The namespace prefixes used in the expression can be defined by clicking the namespaces link in front of the 'Target' input field.
- **Variables for the XQuery Mediator**: Defines values/expressions that could be bound to the dynamic context of the XQuery engine in order to access those variables through the XQuery script.
  - **Variable Type**: The 'type' of the variable must be a valid type defined by the JSR-000225 (XQJ API). The supported types are defined in the Supported Types section.
  - **Variable Name**: The name of the variable should correspond to the name of variable declaration in the XQuery script.
  - **Value Type**: Whether the value of the variable is a static value or an expression.
  - **Value/Expression**: Static value or the expression.
  - **Registry Key**: Key, if the value is retrieved from the registry.
  - **NSEditor**: Defines the namespaces for the prefixes used in the xpath query.

**Examples**

```xml
<xquery key="xquery\example.xq">
  <variable name="payload" type="ELEMENT"/>
</xquery>
```
In the above configuration, XQuery script is in the registry and that script can be picked by key xquery\example.xq. There is a one variable name payload and type as ELEMENT. As there is no expression in the variable definitions, default value - the first child of SOAP Body is used as the value of variable payload. Within XQuery script, you can access this variable by defining declare variable $payload as document-node() external; . Refer sample 390 and 391 for more information.

```xml
<variable name="commission" type="ELEMENT"
key="misc/commission.xml"></variable>
```

A variable definitions that pick a XML resource from the registry using key misc/commission.xml and bind in to XQuery Runtime so that it can be accessed with in XQuery script. Refer sample 391 for more information.

```xml
<variable name="price" type="DOUBLE"
expression="self::node()//m0:return/m0:last/child::text()"
xmlns:m0="http://services.samples/xsd"/>
```

A variable whose value is calculated from current message SOAP Payload using an expression. Here, value is a type of double.

**Example 4.3: Enrich mediation capability**

Enrich mediation capability can process a message based on a given source configuration and then perform the specified action on the message by using the target configuration. It basically gets an OMEElement using the configuration specified in the source and then modify the message by putting it on the current message using the configuration in the target.

**Syntax**

```xml
<enrich>
  <!source [clone=true|false] [type=custom|envelope|body|property|inline]
xpath="" property="" />
  <target [action=replace|child|sibling]
  [type=custom|envelope|body|property|inline] xpath="" property="" />
</enrich>
```

You have the following configuration under the Enrich mediation capability.

- **Source Configuration**
  - Clone : By setting the clone configuration, the message can be cloned or else use as a reference during the enriching. The default value for clone is false. True/False
  - Type : Specifies that the type that the mediator use from the original message to enrich the modified message that pass through the mediator.
  - Custom : Custom XPath value If there are any namespaces involved in the XPath expression, you can specify it in the Namespace Editor.
    - Envelope : Envelope of the original message will be used for enriching.
    - Body : Body of the original message will be used for enriching.
    - Property : Specifies a property.
    - Inline : Specifies an inline XML value

- **Target Configuration**
  - Action : By specifying the action type the relevant action can be applied to outgoing message.
    - Replace : Replace the xml message based on the target type specified on the target configuration.
- Child: Adding as a child of specified target type.
- Sibling: Adding as a sibling of specified target type.

- Type: Specifies that the type of enriching the outgoing message.
  - Custom: Custom XPath value. If there are any namespaces involved in the XPath expression, you can specify it in the Namespace Editor as in the Source Configuration.
  - Envelope: Envelope of the original message will be used for enriching.
  - Body: Body of the original message will be used for enriching.
  - Property: Specifies a property.

Example

```xml
<enrich>
  <!source clone="false" type="envelope" xpath="" property="" />
  <target action="replace" type="body" xpath="" property="" />
</enrich>
```

Example 5: Throttling

You can add throttling to a Service, allowing or denying service access to specified IPs only or limiting max request calls to the Service in a unit of time (this should facilitate the smooth operation of Web services).

Follow the instructions below to add throttling to a service.

1. Sign in. Enter your user name and password to log on to the ESB Management Console.
2. Click on "Main" in the left menu to access the "Manage" menu.
3. In the "Manage" menu, click on "List" under "Web Services."
4. The "Deployed Services" page appears.
5. Select the service for which you want to enable throttling. The "Service Dashboard" page (for that service) appears.
6. In the "Quality of Service Configuration" panel, click "Access Throttling."
7. The "Throttling Configuration" page appears.
8. In the "Enable Throttling" list, select "Yes" from the drop-down menu.
9. The existing throttle configuration appears in the wizard.
10. Click "Add New Entry."

To enter new parameters or modify existing parameters, select "Allow" in the "Access" column.

11. Specify the parameters of a service.

Parameters for Throttling Configuration:

- **Range** - The IP address range or the domain is restricted from accessing the service. Requests from such clients will be restricted based on the specified values.
- **Type** - This indicates the type of Range. It can be IP or DOMAIN. It should be IP if the range is given as a single IP address or a range of IP addresses (for example, 10.100.1.30-10.100.1.60). It should be DOMAIN if the range is given as a domain (for example, *.wso2.com). If you specify configurations types of both IP and DOMAIN, first priority will be given to DOMAIN level configurations.
- **Access**
- Allow - Means that no restriction is applied for that range and all requests are allowed to go in as they come in.
- Deny - Means that access is completely denied for that range.
- Control - If the Access is set to Allow or Deny then Maximum Request Count, Unit Time and Prohibit Time Period parameters are not necessary and the said fields are deactivated. If it is Control, then the specified constraints are applied for that particular range.

- Maximum Request Count (MRC) - If Access is set to Control, it will be the maximum number of requests that are served within the time interval specified by the Unit Time parameter.
- Unit Time (UT) - The time period in milliseconds during which the maximum requests served. This is the number specified by the Maximum Request Count. The throttle starts counting the number of units from the moment it is enabled and the number of requests served within that period.
- Prohibit Time Period (PTP) - If the maximum request count is achieved before the unit time, this is the period during which no more requests are allowed to go in. By setting this value, the unit time slot is altered.

**Control Access example:**

MRC = 50, UT = 50000, PTP = 5000

If 50 requests are arrived within 35000ms (35s) in a particular time period, no more requests are taken in for another 5000ms (5s = PTP).

This time, the UT is altered to 35000ms + 5000ms = 40000ms (40s)

12. Click "Finish." Throttling will be engaged for that particular service.

Functions of Buttons:

- Finish - Click "Finish" to submit the current data. When finished, your throttle configuration will be applied and the page will be redirected to the previous page.
- Reset - Click "Reset" to load the last submitted configuration.
- Default - Click "Default" to load the default throttle configuration. If you want to submit those data, you have to click Finish.
- Clear - Click "Clear" to clear all the text boxes in the user interface.
- Cancel - Click "Cancel" to go to the "Service Dashboard" page.

**Example 6: Load balancing**

A Load balanced endpoint distributes the messages (load) arriving at it among a set of listed endpoints by evaluating the load balancing policy and any other relevant parameters. Policy attribute of the load balance element specifies the load balance policy (algorithm) to be used for selecting the target endpoint. Currently only the roundRobin policy is supported. failover attribute determines if the next endpoint should be selected once the currently selected endpoint has failed, and defaults to true. The set of endpoints among which the load is distributed can be listed under the 'loadBalance' element.

The optional 'session' element makes the endpoint a session affinity based load balancing endpoint. If it is specified, sessions are bound to endpoints in the first message and all successive messages for those sessions are directed to their associated endpoints. Currently there are two types of sessions supported in SAL endpoints. Namely HTTP transport based session which identifies the sessions based on http cookies and the client session which identifies the session by looking at a SOAP header sent by the client with the QName '{http://ws.apache.org/ns/synapse}ClientID'.
Example 7: Failover Endpoints

The Failover Endpoints send messages to the listed endpoints with the following failover behavior. At the start the first listed endpoint is selected as the primary and all other endpoints are treated as backups. Incoming messages are always sent only to the primary endpoint. If the primary endpoint fails, next active endpoint is selected as the primary and failed endpoint is marked as inactive. Thus, it sends messages successfully as long as there is at least one active endpoint among the listed endpoints. Default fail-over endpoint behavior is dynamic fail-over. ESB switches back to the primary endpoint as soon as it is available.

5.3.3.2 Apache Camel

An alternative way to develop data transformation and more in general mediation service is through Apache Camel Route. In this case you can leverage the data transformers provided by Camel. To create a Camel Route using a java class it must extend the class org.apache.camel.builder.RouteBuilder

Camel Route example:

```java
package it.telecomitalia.ictlab.rest.camel;

import it.telecomitalia.ictlab.rest.camel.utils.AuthPredicate;
import it.telecomitalia.ictlab.rest.camel.utils.BaseUrlProcessor;
import it.telecomitalia.ictlab.rest.camel.utils.DbLogProcessor;
import it.telecomitalia.ictlab.rest.camel.utils.XsltURIResolver;
import org.apache.camel.Exchange;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.language.ConstantExpression;
import org.apache.camel.spring.SpringCamelContext;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.web.context.support.XmlWebApplicationContext;

public class PosRestMediator extends RouteBuilder {

    private String POS_SERVICE_ENDPOINT = "http://polaris.cselt.it:9763/PosServiceRest";
```
private String POS_SERVICE_SCHEMA = "posServiceRestSchema.xsd";

@override
public void configure() throws Exception {
    // Register custom XSLTUriResolver in Spring Application Context (registry)
    SpringCamelContext camelContext = (SpringCamelContext)getContext();
    XmlWebApplicationContext context = (XmlWebApplicationContext)camelContext.getApplicationContext();
    DefaultListableBeanFactory beanFactory = (DefaultListableBeanFactory) context.getBeanFactory();
    try {
        beanFactory.registerSingleton("XsltURIResolver", new XsltURIResolver());
    } catch (Exception e) {
        e.printStackTrace();
    }
    //PosService
    from("servlet:///posService?matchOnUriPrefix=true")
        .streamCaching()
        .to("log:logger")
        .to("direct:validatePosService")
        .choice()
            .when(new AuthPredicate("PosService"))
                .process(new DbLogProcessor())
                .removeHeader("Authorization")
                .to(POS_SERVICE_ENDPOINT + "/rest?bridgeEndpoint=true&throwExceptionOnFailure=false")
                .to("log:logger")
                .to("direct:validatePosService")
                .to("log:logger")
            .otherwise()
                .to("direct:return401_3")
        .end();

    //WADL
    from("servlet:///posService/wadl")
        .to(POS_SERVICE_ENDPOINT + "/rest/application.wadl?bridgeEndpoint=true")
.setHeader("importSchema", new ConstantExpression(POS_SERVICE_SCHEMA))
    .process(new BaseUrlProcessor("camelServices/posService"))
    .to("xslt:wadl.xsl");

    //Schema import
    from("servlet:///posService/" + POS_SERVICE_SCHEMA)
    .to(POS_SERVICE_ENDPOINT + "/" + POS_SERVICE_SCHEMA +"?bridgeEndpoint=true")
    .setHeader("Content-Type", new ConstantExpression("application/xml"));

    //HTML docs
    from("servlet:///posService/doc")
    .to(POS_SERVICE_ENDPOINT + "/rest/application.wadl?bridgeEndpoint=true")
    .setHeader("importSchema", new ConstantExpression(POS_SERVICE_SCHEMA))
    .process(new BaseUrlProcessor("camelServices/posService"))
    .to("xslt:wadl.xsl")
    .to("xslt:wadl_documentation.xsl?transformerFactoryClass=net.sf.saxon.TransformerImpl&uriResolver=XsltURIResolver");

    //PosService XSD validation
    from("direct:validatePosService")
    .choice()
    .when(header("Content-Type").isEqualTo("application/xml"))
    .to("log:validationLogger")
    .to("validator:" + POS_SERVICE_ENDPOINT + "/" + POS_SERVICE_SCHEMA)
    .to("log:validationLogger")
    .end();

    //Prepare 401 unauthorized response
    from("direct:return401_3")
    .setHeader(Exchange.HTTP_RESPONSE_CODE, new ConstantExpression("401"))
    .setHeader("WWW-Authenticate", new ConstantExpression("Basic realm="WSO2\""))
5.3.4 Back-end services exposed
The Mediator GE exposes by default some back-end service.

5.3.4.1 GeoFencing service
The GeoFencing service allows to locate mobile devices. The GeoFencing service is composed by:

- the server part as a SOAP Web Service, available through the Mediator GE at https://<mediator instance host>:8243/services/fiwareGeofencerProxy
- the Location Service, that must be installed on the mobile device. At the moment the Location Service is available only for Android Smartphones: https://forge.fi-ware.org/frs/download.php/1033/LocationService.apk

Through the installed LocationService the user can register the mobile device i.e. enable the mobile device to be localized through the GeoFencing service.

A registered mobile device can be localized through calls to the GeoFencing SOAP Web Service. GeoFencing SOAP Web Service is described by the WSDL at: http://<mediator instance host>:8280/services/fiwareGeofencerProxy?wsdl Both single shot and periodic localizations are supported.

- Single shot location returns the localization data in the payload of the response message.
- Periodic localizations requires the user to specify a URI where the periodic localization data will be sent: the URI must be specified in the "NotifyTo" request field.

The service offers also: **Presence in area detection**: the service can communicate the presence in specified areas of registered mobile devices. The user can specify the areas and events (enter area, exit area) that he is interested to be notified of. **Proximity**: the service can communicate the distance between different registered mobile devices.
6  Registry - User and Programmer Guide

6.1  Introduction
This document describes the necessary steps to develop a software application or a user interface which makes use of the Registry back-end functionality. The Registry API is based on REST principles and generally accepts and returns JSON encoded messages. Since REST is independent from a concrete programming language, you just have to know how to make an HTTP request in the programming language of your choice.

6.1.1  Background and Detail
This User and Programmers Guide relates to the Registry GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

6.2  User Guide
Since the Registry is a Generic Enabler which provides pure backend functionality to other applications (e.g. Generic Enablers or end user facing applications), we do not distinguish between the User and Programmers Guide. Please refer to the Programmers Guide section for more information.

6.3  Programmer Guide
To give a feeling of how the Registry works and how to interact with the system lets take a look at some examples, realized with the command line tool cURL and in Java. 'cURL' is a command which can be used to perform any kind of HTTP operation - and therefore is also usable for the Registry. The library 'libcurl' can be integrated in C programs as well.

6.3.1  Accessing the Registry with cURL
6.3.1.1  Create a registry entry with cURL
This example shows how to create a registry entry with the command line tool 'cURL':

Create a Resource and save it to a file named test.json

```json
{
  "type": "Person",
  "firstName": "Joe",
  "lastName": "Random",
}
```

send the request to the server

```bash
```

You should obtain the following result
6.3.1.2 Get a registry entry with cURL


Yields the following output:

* About to connect() to localhost port 5000 (#0)
* Trying 127.0.0.1... connected
> GET /de/acme/Joe%20Random HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libIdn/1.23 librtmp/2.3
> Host: localhost:5000
> Accept: */*
> 
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Last-Modified: Invalid Date
< Content-Type: application/json; charset=utf-8
< Content-Length: 67
< Set-Cookie: connect.sid=I0InZoT3cPRjNTaFoWkhloHy.bsnQXxyu9V5zTwA4leMkMGMa3IniYPBWcnwm4b4HZbA; path=/; expires=Thu, 04 Oct 2012 13:29:25 GMT; httpOnly
< Date: Thu, 04 Oct 2012 09:29:25 GMT
< Connection: keep-alive
<
6.3.1.3 Get all registry entries of a common basename

Suppose there are multiple entries with the same basename it yields the following output:

```bash
```

6.3.1.4 Delete a registry entry with cURL

Sending the request to the server

```bash
```
You should obtain the following result:

* About to connect() to localhost port 5000 (#0)
* Trying 127.0.0.1... connected
> DELETE /registry/de/acme/Joe%20Random HTTP/1.1
> User-Agent: curl/7.23.1 (i386-pc-win32) libcurl/7.23.1 zlib/1.2.5
> Host: localhost:5000
> Accept: */*
>
< HTTP/1.1 204 No Content
< Date: Wed, 11 Jul 2012 12:13:26 GMT
<

In this case the HTTP DELETE operation is used. The Registry uses the Create Retrieve Update Delete (CRUD) operations which map almost to the HTTP verbs PUT, GET, POST and DELETE.
7  Repository - User and Programmer Guide

7.1  Introduction
This document describes the necessary steps to develop a software application or a user interface which makes use of the Repository backend functionality. The Repository API is based on REST principles and generally returns XML or JSON encoded responses. Since REST is independent of a concrete programming language, you just have to know how to make an HTTP request in the programming language of your choice.

7.1.1.1  Background and Detail
This User and Programmers Guide relates to the Repository GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

7.2  User Guide
Since the Repository is a Generic Enabler which provides pure backend functionality to other applications (e.g. Generic Enablers or end user facing applications), we do not distinguish between the User and Programmers Guide. Please refer to the Programmers Guide section for more information.

7.3  Programmer Guide
To give you a feeling of how the Repository works and how you can interact with the system let us take a look at some examples, realized with the command line tool cURL and in Java. 'cURL' is a command which can be used to perform any kind of HTTP operation - and therefore is also usable for the Repository. The library libcurl enables the integration in C programs as well.

7.3.1  Accessing the Repository with cURL
7.3.1.1  Create an Offering with cURL
This example shows how to create a resource with the command line tool 'cURL':

- Create a Resource and save it to a file named test.txt
- send the request to the server

```
```

- You should obtain the following message

```
* About to connect() to localhost port 8080 (#0)
* Trying 127.0.0.1... connected
> PUT /FiwareRepository/v1/collectionA/collectionB/ResourceName HTTP/1.1
> User-Agent: curl/7.23.1 (i386-pc-win32) libcurl/7.23.1 zlib/1.2.5
```
7.3.1.2  **Delete a resource with cURL**

* send the request to the server

```bash
```

* You should obtain the following message

```
* About to connect() to localhost port 8080 (#0)
* Trying 127.0.0.1... connected
> DELETE /FiwareRepository/v1/collectionA/collectionB/ResourceName HTTP/1.1
> User-Agent: curl/7.23.1 (i386-pc-win32) libcurl/7.23.1 zlib/1.2.5
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 204 No Content
< Server: Apache-Coyote/1.1
< Date: Wed, 11 Jul 2012 12:13:26 GMT
<
* Connection #0 to host localhost left intact
* Closing connection #0
```

In this case the HTTP DELETE operation is used. The Repository uses the Create Retrieve Update Delete (CRUD) operations which map almost to the HTTP verbs PUT, GET, POST and DELETE.
7.3.2 Accessing the Repository with Java

7.3.2.1 Create a Resource

- The following functions demonstrates how you can upload a new resource, or update an existing resource on the repository. `repositoryURL` is the base URL of the Repository, `resourceId` is the path to the resource, `file` is the file you want to upload as resource.

```java
public Boolean insertResourceContent(String repositoryURL, String resourceId, String file){
    try {
        HttpClient httpclient = new DefaultHttpClient();
        HttpPost httppost = new HttpPost(repositoryURL+resourceId);

        MultipartEntity reqEntity = new MultipartEntity();
        File f = new File(file);
        FileBody bin = new FileBody(f);

        String mimeType = new String(new MimetypesFileTypeMap().getContentType(f));
        StringBody comment = new StringBody(f.getName());

        reqEntity.addPart("filename", comment);
        reqEntity.addPart("mimeType", mimeType);
        reqEntity.addPart("filedata", bin);

        httppost.setEntity(reqEntity);
        HttpResponse response;
        response = httpclient.execute(httppost);
        HttpEntity resEntity = response.getEntity();

        return true;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
       
```
7.3.2.2 **Delete a Resource**

- The following function demonstrates how you can delete a resource which is stored in the Repository. `repositoryURL` is the base URL of the Repository, `resourceID` is the path to the resource.

```java
public Boolean deleteResource(String repositoryURL, String resourceID){
    try {
        ClientRequest request = new ClientRequest(repositoryURL+resourceID);
        request.accept("application/xml");
        ClientResponse<String> response = request.delete(String.class);
        ClientUtil.visualize(request, response, "Delete Resource");

        if(response.getStatus() == 200){
            return true;
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}
```

7.3.3 **Retrieving Meta Data information about a Resource in different formats**

HTTP content negotiation allows the client to choose the appropriate data format for retrieving meta information about a resource or a collection. Besides RDF, XML, TURTLE, and JSON the Repository also supports human readable output formats using HTML rendering (`text/html` accept header) including hyperlinked representation and formatted text.

7.3.3.1 **HTML Representation**

- Request URL: http://[REPOSITORY_URL]/testCollection/
- Accept Header: text/html, application/x-ms-application
- Result:
7.3.3.2  Test Representation

- Request URL: http://[REPOSITORY_URL]/testCollection/
- Accept Header: text/plain
- Result:

Collection: testCollection
Creation Date: Thu Mar 21 10:46:39 CET 2013

Collections:

<table>
<thead>
<tr>
<th>Collection Id</th>
<th>Creation Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>testCollection/collectionA</td>
<td>Thu Mar 21 10:47:53 CET 2013</td>
</tr>
<tr>
<td>testCollection/collectionB</td>
<td>Thu Mar 21 10:47:53 CET 2013</td>
</tr>
<tr>
<td>testCollection/</td>
<td>Thu Mar 21 10:47:53 CET 2013</td>
</tr>
</tbody>
</table>
7.3.3.3 **JSON Representation**

- **Request URL:** http://[REPOSITORY_URL]/testCollection/
- **Accept Header:** application/json
- **Result:**

```json
{
    "resources": [
      {
        "resourceId": "testCollection/testResource1",
        "creationDate": "Thu Mar 21 10:46:39 CET 2013",
        "modificationDate": "Thu Mar 21 10:46:39 CET 2013",
        "filename": "filename",
        "mimeType": "application/rdf+xml"
      },
      {
        "resourceId": "testCollection/testResource2",
        "creationDate": "Thu Mar 21 10:47:53 CET 2013",
        "modificationDate": "Thu Mar 21 10:47:53 CET 2013",
        "filename": "filename",
        "mimeType": "plain/text"
      },
      {
        "resourceId": "testCollection/testResource3",
        "creationDate": "Thu Mar 21 10:47:53 CET 2013",
        "modificationDate": "Thu Mar 21 10:47:53 CET 2013",
        "filename": "filename",
        "mimeType": "text/turtle"
      },
      {
        "resourceId": "testCollection/testResource4",
        "creationDate": "Thu Mar 21 10:47:53 CET 2013",
        "modificationDate": "Thu Mar 21 10:47:53 CET 2013",
        "filename": "filename",
        "mimeType": "application/rdf+xml"
      }
    ]
}
```
Future Internet Core Platform

"name": "",
"content": null,
"collection": null,
"contentMimeTpye": "application\/rdf+xml",
"contentFileName": "filename",
"contentUrl": "",
"id": "testCollection\//testResource1",
"creationDate": 1363859199839,
"creator": "",
"modificationDate": 1363859199839
},
{
"name": "",
"content": null,
"collection": null,
"contentMimeTpye": "plain\//text",
"contentFileName": "filename",
"contentUrl": "",
"id": "testCollection\//testResource2",
"creationDate": 1363859273515,
"creator": "",
"modificationDate": 1363859273515
},
{
"name": "",
"content": null,
"collection": null,
"contentMimeTpye": "text\//turtle",
"contentFileName": "filename",
"contentUrl": "",
"id": "testCollection\//testResource3",
"creationDate": 1363859273535,
"creator": "",
"modificationDate": 1363859273535
},
{
"name": "",
"content": null,
null,
7.3.3.4 RDF/XML Representation

- Request URL: http://[REPOSITORY_URL]/testCollection/
- Accept Header: application/rdf+xml
- Result:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://purl.org/dc/terms/" >
    <rdf:Description
        rdf:about="http://localhost:7080/FiwareRepository/v1/testCollection"
        >
        <j.0:date>Thu Mar 21 10:46:39 CET 2013</j.0:date>
    </rdf:Description>
    <rdf:Description
        rdf:about="http://localhost:7080/FiwareRepository/v1/testCollection#collections/">
        <rdf:_3
            rdf:resource="http://localhost:7080/FiwareRepository/v1/testCollection/"/>
        <rdf:_2
        <rdf:_1
        <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
    </rdf:Description>
</rdf:RDF>
```
7.3.3.5 **Turtle Representation**

- Request URL: http://[REPOSITORY_URL]/testCollection/
- Accept Header: text/turtle
- Result:

```turtle
@prefix dc: <http://purl.org/dc/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://localhost:7080/FiwareRepository/v1/testCollection> a <http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag> ;

<http://localhost:7080/FiwareRepository/v1/testCollection#collections/> a <http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_1> .

<http://localhost:7080/FiwareRepository/v1/testCollection/collectionA> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2> .

<http://localhost:7080/FiwareRepository/v1/testCollection/collectionB> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_3> .

```
7.3.3.6  **N-Triple Representation**

- Request URL: `http://[REPOSITORY_URL]/testCollection/`
- Accept Header: `text/n3`
- Result:

```
<http://localhost:7080/FiwareRepository/v1/testCollection#resources/>
  a  <http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_1>

<http://localhost:7080/FiwareRepository/v1/testCollection/testResource1> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2>

<http://localhost:7080/FiwareRepository/v1/testCollection/testResource2> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_3>

<http://localhost:7080/FiwareRepository/v1/testCollection/testResource3> ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#_4>

```
7.3.3.7  XML Representation

- Request URL: http://[REPOSITORY_URL]/testCollection/
- Accept Header: application/xml
- Result:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection id="testCollection" xmlns:atom="http://www.w3.org/2005/Atom">
  <creationDate>2013-03-21T10:46:39.837+01:00</creationDate>
  <creator/>
  <collections>
    <collections id="testCollection/collectionA">
      <creationDate>2013-03-21T10:47:53.552+01:00</creationDate>
      <creator/>
      <collections/>
    </collections>
    <resources/>
  </collections>
  <collections id="testCollection/collectionB">
    <creationDate>2013-03-21T10:47:53.566+01:00</creationDate>
    <creator/>
    <collections/>
    <resources/>
  </collections>
  <collections id="testCollection/">
    <creationDate>2013-03-21T10:47:53.575+01:00</creationDate>
    <creator/>
    <collections/>
    <resources/>
  </collections>
</collection>
```
</collections>
</collections>
</resources>

<resources id="testCollection/testResource1">
  <creationDate>2013-03-21T10:46:39.839+01:00</creationDate>
  <modificationDate>2013-03-21T10:46:39.839+01:00</modificationDate>
  <contentFileName>filename</contentFileName>
  <contentMimeType>application/rdf+xml</contentMimeType>
  <name/>
</resources>

<resources id="testCollection/testResource2">
  <creationDate>2013-03-21T10:47:53.515+01:00</creationDate>
  <modificationDate>2013-03-21T10:47:53.515+01:00</modificationDate>
  <contentFileName>filename</contentFileName>
  <contentMimeType>plain/text</contentMimeType>
  <name/>
</resources>

<resources id="testCollection/testResource3">
  <creationDate>2013-03-21T10:47:53.535+01:00</creationDate>
  <modificationDate>2013-03-21T10:47:53.535+01:00</modificationDate>
  <contentFileName>filename</contentFileName>
  <contentMimeType>text/turtle</contentMimeType>
  <name/>
</resources>

<resources id="testCollection/testResource4">
  <creationDate>2013-03-21T10:47:53.545+01:00</creationDate>
  <modificationDate>2013-03-21T10:47:53.545+01:00</modificationDate>
  <contentFileName>filename</contentFileName>
  <contentMimeType>application/rdf+xml</contentMimeType>
  <name/>
</resources>
8 RSS - User and Programmer Guide

8.1 Introduction
This page contains the User and Programmer Guide of the RSS GE (Revenue Settlement and Sharing Generic Enabler) and its functionalities. RSS GE has been built following the specification done at:

- FIWARE.OpenSpecification.Apps.RSSRest
- FIWARE.OpenSpecification.Apps.RSS.RSSModels

The enabler offers the following services:

- REST service to receive CDRs (Charging Data Records)
- REST service to control the expenditure limits for users
- A Web Interface service for RSS administrators
  - Management of settlement process, files and transactions
  - Management of RS models
  - Management of stores and application providers
  - Access to graphical reports
- A Web Interface service for Store administrators
  - Management of their settlement process, files and transactions
  - Management of their RS models
  - Management of their application providers
  - Access to their graphical reports

The APIs offered are based on REST principles. Since REST is independent of a concrete programming language, it is only necessary to know how to make an HTTP request in the programming language chosen.

8.2 User Guide
RSS is a Generic Enabler which provides pure backend functionality to other applications (e.g. Generic Enablers or end user facing applications). However, two web interfaces are offered in order RSS administrators and registered stores can interact with the back-end functionality:

- RSS Private Back End Interface: All operations regarding RSS can be accessed by command line. However, in order to make these tasks more accessible, a simple web interface for RSS and stores administrators has been developed to make them easier.
- Pentaho Business Intelligence (BI) Server Interface: Through this interface an administrator can access to the functionality of generating reports regarding the CDRs received such as Top applications Report. Besides, registered stores can also access to this functionality but restricted to the information related to the store. More information about Pentaho BI Server can be consulted at:

  [http://community.pentaho.com/](http://community.pentaho.com/)

8.2.1 RSS Back End Interface
This WEB Interface allows a RSS or a Store administrator to perform its core operations.
It is located at: [http://host:port/fiware-rss/](http://host:port/fiware-rss/). It will appear a page with a link to an external page (IDM account portal) where the user needs to have an account and permission to access to the portal. This link is the place where getting an **IDM account**.

**RSS - Settlement**

It is necessary being logged to access to the page. Please login in accounting page. [Login](#)

After clicking in the link, the IDM page will appear:

After filling the correct login and password in, the user will be redirected to the main page of the RSS private backend interface. The following first figure shows the main page for the RSS administrator, and the second one for the Store administrator.
As can be seen in the above figures, the web interface for the Store administrator is the same than the RSS one, but without having the option to create new stores or delete transactions. Moreover, for the Store administrator one, only the affected settlement files, transactions, RS models, providers and reports are managed.

The next sections explain each of the options in the administration portal.

8.2.1.1 **RSS API**

This option shows the services offered by the RSS API. The following text will appear on the WEB:

<table>
<thead>
<tr>
<th>Available SOAP services:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available RESTful services:</td>
</tr>
<tr>
<td>Endpoint address: <a href="http://host:port/fiware-rss/rss/cdrs">http://host:port/fiware-rss/rss/cdrs</a></td>
</tr>
<tr>
<td>WADL: <a href="http://host:port/fiware-rss/rss/cdrs?_wadl">http://host:port/fiware-rss/rss/cdrs?_wadl</a></td>
</tr>
</tbody>
</table>

8.2.1.2 **Launch settlement**

This option launches the process that generates the revenue sharing among providers of the CDRs stored in the database. There are two ways of working:

- If no parameters are specified, the settlement is calculated for the current month.
- If From and To parameters are specified, this is the period to calculate the settlement.
- Provider. Identifies the provider to whom the settlement will be launched. If no provider is specified, the settlement will be launched for all the providers or those managed by the Store.

It generates an excel file containing the information needed to share revenues between the different providers. If everything goes OK, the following message will be displayed:
This process could require an amount of time of one or two minutes. After that, we will be able to retrieve the files generated with the following option.

8.2.1.3 **View files**

This option shows the existing revenue sharing files and allows the RSS or Store administrator to download them.

As an example, the following information will be showed:

```
RSS - Settlement

RSS Files
    generic_report_conwet_FI-WARE_FS_2013-02_2014-02.csv
    conwet-FI-WARE-FS-2013-02-TO-2014-02.xls
```

From here, the administrator can download the “.csv” or “.xls” file containing the specific revenue sharing information.

8.2.1.4 **View transactions in database**

This option shows the existing transactions inside the RSS Database for every provider or for the corresponding Store providers.

An example of the information returned is:

```
RSS - Settlement

<table>
<thead>
<tr>
<th>Provider ID</th>
<th>User ID</th>
<th>Trx Type</th>
<th>App Id</th>
<th>Request Time</th>
<th>Ref Code Amount</th>
<th>Tax Amount</th>
<th>Total Amount</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conwet</td>
<td></td>
<td></td>
<td>Smart City Lights</td>
<td>2013-12-01 13:51:45</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>Single payment: 4.0 EUR</td>
</tr>
<tr>
<td>conwet</td>
<td></td>
<td></td>
<td>Santeder City Lights</td>
<td>2013-12-01 13:26:31</td>
<td>57</td>
<td>3.0000</td>
<td>6.0000</td>
<td>Single payment: 1.0 EUR</td>
</tr>
<tr>
<td>conwet</td>
<td></td>
<td></td>
<td>Santeder City Lights</td>
<td>2013-12-01 13:26:31</td>
<td>41</td>
<td>3.0000</td>
<td>6.0000</td>
<td>Single payment: 1.0 EUR</td>
</tr>
<tr>
<td>conwet</td>
<td></td>
<td></td>
<td>Santeder City Lights</td>
<td>2013-12-01 13:26:31</td>
<td>98</td>
<td>6.0000</td>
<td>6.0000</td>
<td>Single payment: 1.0 EUR</td>
</tr>
<tr>
<td>conwet</td>
<td></td>
<td></td>
<td>Santeder City Lights</td>
<td>2013-12-01 13:54:42</td>
<td>35</td>
<td>3.0000</td>
<td>6.0000</td>
<td>Single payment: 1.0 EUR</td>
</tr>
</tbody>
</table>
```
8.2.1.5 **View RS models in database**
This option shows the existing Revenue Sharing Models inside the RSS Database. In case of a Store administrator, only the RS models for its providers appear.
An example of the information returned is:

![RSS - Settlement](image)

<table>
<thead>
<tr>
<th>Provider ID</th>
<th>Revenue Share %</th>
<th>Product Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoNWeT</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>UPM</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>convet</td>
<td>80</td>
<td>applications</td>
</tr>
<tr>
<td>newProvider</td>
<td>80</td>
<td>Games</td>
</tr>
<tr>
<td>newProvider</td>
<td>75</td>
<td>Films</td>
</tr>
<tr>
<td>newProvider</td>
<td>60</td>
<td>music</td>
</tr>
</tbody>
</table>

8.2.1.6 **View Providers in database**
This option shows the existing providers inside the RSS Database. In case of a Store administrator, only its providers appear.
An example of the information returned is:

![RSS - Settlement](image)

<table>
<thead>
<tr>
<th>Provider ID</th>
<th>Provider Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>generic AppProvider</td>
</tr>
<tr>
<td>CoNWeT</td>
<td>CoNWeT</td>
</tr>
<tr>
<td>UPM</td>
<td>UPM</td>
</tr>
<tr>
<td>convet</td>
<td>convet</td>
</tr>
<tr>
<td>newProvider</td>
<td>newProvider</td>
</tr>
<tr>
<td>providerTestId</td>
<td>providerTestId</td>
</tr>
<tr>
<td>wstore-test</td>
<td>WStore-test</td>
</tr>
</tbody>
</table>

8.2.1.7 **Create RS Models in database**
This option allows the creation of providers into the database. Two parameters are required and a third one optional:
- Provider: It is the identifier of the provider that the model is created for. In case of a Store administrator, only its providers appears in this field.
• Revenue %: Percentage of revenue between Application provider and the Store.
• Product Class: This parameter is optional. One provider can have different revenue percentages depending on the kind of good is being selling (for example: application, music).

If everything goes OK, the following message will be displayed:

8.2.1.8 Register Providers in database
This option allows the registration of providers into the database. Two parameters are required and a third one optional:
• Provider Id: It is the identifier that must be used in requests regarding itself.
• Provider Name: It is the common name given to the provider.
• Store (optional): A provider could be associated to a store, that offers products from different providers. This option doesn't appear in case of Store administrators as they only are able to create providers for their stores.

If everything goes OK, the following message will be displayed:
8.2.1.9 **Register Store in database**

This option allows the creation of stores into the database. This option is only accessible for RSS administrators. Two parameters are required:

- Store email provider: It is the identifier that will be used for authentication and administration purposes.
- Store Name: It is the common name given to the store.

If everything goes OK, the following message will be displayed:

![Store Created](image1)

After registering the store in the database it is needed to setup this new store, so its administrator can access the admin portal and access the reports as detailed in the **Set Up new Store** section.

8.2.1.10 **Delete transactions of provider**

This option allows the RSS administrator to delete all existing transactions of any provider he wants to. This feature is only accessible for RSS administrators.

In the form fields, it is necessary to select the provider whose transactions will be deleted. After deleting the transactions, the following message will be shown:

![Deleting Done](image2)
8.2.1.11 **View reports**

This option allows the RSS administrator or the Store administrator to access to Pentaho BI server interface, where different reports could be generated.

8.2.2 Set Up new Store

Previously a Store can use the RSS services, it is necessary to perform a series of operations:

- It is necessary to have an Identity Manager (IDM enabler) account. This account will receive privileges from RSS administrator, to be able to interact with RSS.
- It is necessary to register the Store in the database, using as identifier the email that the Store has associated to the above IDM account. After this step, the Store administrator will have access to RSS Back End Interface (explained in the previous section). Using this interface, the Store will be able to register its providers, that will send CDRs to RSS.
- Finally, in order to allow the Store administrator to generate reports by using Pentaho BI Server with its administrator email (the same one in the previous step), it is necessary to register it as an authorised user in Pentaho BI Server. To do it, follow the steps detailed in the Installation and administration guide using as username the Store administrator email and a arbitrary password, that the RSS administrator will send back to Store administrator.

8.2.3 Pentaho BI Server Interface

This is the Web interface where the Pentaho reports are available. It is located at:

```
http://host:port/pentaho/Login
```

When accessing the above URL, the Pentaho BI Server login page must appear:

![Pentaho User Console](image)

Use the login data according your credentials as RSS or Store administrator.

And the main page will appear. Click on "FI-WARE reports" link on the lower left menu, and the 4 available reports will appear in the lower left panel:
In the next sections, each report is explained.

8.2.3.1 *Top Applications Report*

This report shows the information of the most purchased applications in a period of time.

From the available reports, choose "Top applications" by double-clicking on it. A web form will appear where it is required to fill in the following fields:

- **Provider**: Service provider taken into account to generate the report.
- **Date from**: Starting date of the transactions to report.
- **Date to**: Final date of the transactions to report.
- **Ranking criteria**: Sort order criteria (by: Incomes, Charges amount, Refunds amount, Unique users, Transactions, Charge transactions, Refund transactions or Failed transactions).
- **Top limit**: Maximum number of applications (if achieved) to show in the report.
- **Net or Gross evaluation**: Money to take into account with or without taxes (if exist).

As an example, fill the web form with the following information:

- **Provider**: All
- **Date from**: 01/01/2013 00:00:00
- **Date to**: 30/11/2013 00:00:00
- **Ranking criteria**: Incomes
- **Top limit**: 25
- **Net or Gross evaluation**: Gross
Results obtained:

A pdf file reporting the list of services that have generated more incomes in the specified period. The information is shown in a graphical and tabular ways containing the amount of incomes, number of transactions, number of users, summary data and comparative graphs.
8.2.3.2 Transactions Report

This report shows the information about the transactions inserted during a period of time.

From the available reports, choose "Transactions" by double-clicking on it. A web form will appear where it is required to fill in the following fields:

- **Provider**: Service provider taken into account to generate the report.
- **Date from**: Starting date of the transactions to report.
- **Date to**: Final date of the transactions to report.
- **User Id**: The customer identifier.
  - In this field, it is possible to used patterns ('_' any character or '%' a string of any length). This is made in order to allow searches for more than one user identifier.
- **Grouping by**: The way transactions will be grouped (by: Operation, User Id, Status, Request description, provider or none)
- **Order By**: The way transactions will be sorted (by: Operation, Status, Transaction Id, User Id, Gross Income, Net Income, Request description, Request date, Elapsed time (ms) or Provider)

As an example fill in the following information:

- **Provider**: All
- **Date from**: 01/01/2013 00:00:00
- **Date to**: 30/11/2013 00:00:00
- **User Id**: Left empty.
- **Grouping by**: None
- **Order By**: Operation
Results obtained:
A pdf file reporting the every related information of the transactions stored during the specified period.

8.2.3.3 Summary Report
This report shows a summary of the evolution of the arriving transactions during a period alone or comparing it with another period of time.

From the available reports, choose “Summary” by double-clicking on it. A web form will appear where it is required to fill in the following fields:

- compare two date ranges?: If you want to compare two date ranges. If not, only the first period of time is required.
- show last transaction data?: If you want to show also the individual information of the last transaction.
- Provider: Service provider taken into account to generate the report.
- Date from: Starting date of the transactions to report.
- Date to: Final date of the transactions to report.
- Compare with Date from: Starting date of the transactions to compare to.
- Date to: Final date of the transactions to compare to.

As an example, fill the web form with the following information:
- compare two date ranges?: Yes
- show last transaction data?:Yes
- Provider: All
- Date from: 01/01/2013 00:00:00
- Date to: 31/12/2013 00:00:00
- Compare with Date from: 01/12/2013 00:00:00
- Date to: 31/12/2013 00:00:00

Results obtained:
A pdf file reporting the daily evolution of the number of transactions during the specified period for every service provider (or only the service providers registered in the store): daily evolution of the number of transactions, distribution by transaction type, summary of the transactions and the last transaction details.
8.2.3.4 **Price Interval Report**

This report shows a comparison of the distribution of the paying amount of different charges and refunds in two time periods.

This distribution is shown in terms of maximum and minimum values and the three quartiles. The quartiles of a ranked set of data values are the three points that divide the data set into four equal groups, each group comprising a quarter of the data.

The minimum value is shown as a red circle and the maximum value is shown as a green circle. Between them, quartiles 1 and 3 are tagged as Q1 and Q3, and the second quartile (or median) is represented by a red number. The 50% most representative price interval is the one between Q1 and Q3. Therefore, it is represented by a thicker blue line. Example:

![Price Interval Distribution Diagram]

From the available reports, choose "Price Interval" by double-clicking on it. A web form will appear where it is required to fill in the following fields:

- **Provider**: Service provider taken into account to generate the report.
- **Date from**: Starting date of the first time period to report.
- **Date to**: Final date of the first time period to report.
• Compare with Date from: Starting date of the second time period to report.
• Date to: Final date of the second time period to report.
• Min representation limit: Minimum representation price value for the report (optional)
• Max representation limit: Maximum representation price value for the report (optional)
• Net or Gross evaluation?: Show quantities with or without taxes (if possible)
• Show legends: Show information regarding the axes of the graphics.

As an example, fill the web form with the following information:

• Provider: All
• Date from: 01/01/2013 00:00:00
• Date to: 31/12/2013 00:00:00
• Compare with Date from: 01/12/2013 00:00:00
• Date to: 31/12/2013 00:00:00
• Min representation limit: left empty
• Max representation limit: left empty
• Net or Gross evaluation?: Gross
• Show legends?: Yes

Results obtained: A pdf file reporting the settlement price interval during the specified periods for every service provider (or only the service providers registered in the store).
8.3 Programmer Guide

RSS GE offers different API for each of the available functionalities. Next sections introduce each API, showing what is offered and the way a developer can use them. In this way, some examples are shown using the linux command cURL.

‘cURL’ is a command which can be used to perform any kind of HTTP operation - and therefore is also usable for the RSS.

One important issue to highlight is the use of authentication security header. If security is enabled (see Installation and administration guide), it is mandatory to include the X-Auth-Token HTTP Header that will contain a security authentication token obtaining from IDM account System.

8.3.1 RSS API REST service

This API exposes the functionality that a partner (usually a store) will use to send its CDRs (Charging Data Record) to the RSS enabler for their processing.

The RSS API is located at:

```
http://host:port/fiware-rss/rss/cdrs
```

There is a way of obtaining the services offered by the API, using the following URL in a browser:

```
http://host:port/fiware-rss/rss/cdrs?_wadl
```

The only request accepted by the service is:
HTTP POST requests
*Content Type: application/xml

The following sections show the structure of the information required in the body of the request and some examples.

8.3.1.1 CDR fields detail

The body sent in a request to RSS API is a group of CDRs to be stored in the database. The structure is as follows:

```xml
<cdrs>
  <cdr>
    <cdr_type>Charge</cdr_type>
    <id_service_provider>TestServiceProvider</id_service_provider>
    <id_application>TestApplication</id_application>
    <id_event>Pay per use event</id_event>
    <id_correlation>8407_1474276322_20130213T09:11:56</id_correlation>
    <purchase_code>0</purchase_code>
    <parent_app_id>TestApplication</parent_app_id>
    <product_class>PaaS</product_class>
    <description>ASM fee per call</description>
    <cost_currency>1</cost_currency>
    <cost_units>1,0000</cost_units>
    <tax_currency>1</tax_currency>
    <tax_units>0,0000</tax_units>
    <cdr_source>1</cdr_source>
    <id_operator>1</id_operator>
    <id_country>1</id_country>
    <time_stamp>2013/02/13 09:11:56</time_stamp>
    <id_user>customer</id_user>
  </cdr>
</cdrs>
```

The description of each field is:
- **cdr_type**: (optional) possible values: "Charge" (default), "Refund".
- **id_service_provider**: Service Provider id.
- **id_application**: Application id.
- **id_event**: Event id.
- **id_correlation**: CDR id, must be unique.
- **purchase_code**: In "Charge" CDRs, the purchase code must be different for each charge. In "Refund" CDRs, the purchase code must be equal to the purchase code of the original charge CDR which is refunded.
- **parent_app_id**: Parent application id.
- **product_class**: this is field will be used by revenue sharing system as a parameter to retrieve the revenue model to apply for this transactions. As an example a value could be: SaaS.
- **description**: description of the transaction.
- **cost_currency**: currency of the transaction (see currency ids in the following table).
- **cost_units**: cost amount without taxes.
- **tax_currency**: tax currency of the transaction (see currency ids in the following table).
- **tax_units**: taxes amount.
- **cdr_source**: The system is able to receive CDR from different sources but for the current situation there is only one so the value must be 1.
- **id_operator**: must be 1.
- **id_country**: see country ids in the following table.
- **time_stamp**: time of the transaction.
- **id_user**: User id.
- **refund_reason**: (optional) Text description of the refund reason.

Note: In "Refund" CDRs the units aren't needed because the amounts refunded are the amounts of the original "Charge" CDR.

### 8.3.1.1.1 Country and currency values

The country id, cost_currency and tax_currency must be a numeric value as shown in the following table.

<table>
<thead>
<tr>
<th>id_country</th>
<th>Name</th>
<th>Country code (ISO 3166)</th>
<th>id_currency</th>
<th>Currency description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Spain</td>
<td>ES</td>
<td>1</td>
<td>Euro</td>
</tr>
<tr>
<td>2</td>
<td>Great Britain</td>
<td>GB</td>
<td>2</td>
<td>Esterlina</td>
</tr>
<tr>
<td>3</td>
<td>Germany</td>
<td>DE</td>
<td>1</td>
<td>Euro</td>
</tr>
<tr>
<td>4</td>
<td>Mexico</td>
<td>MX</td>
<td>5</td>
<td>Peso mexicano</td>
</tr>
<tr>
<td>5</td>
<td>Chile</td>
<td>CL</td>
<td>6</td>
<td>Peso chileno</td>
</tr>
<tr>
<td>6</td>
<td>Argentina</td>
<td>AR</td>
<td>4</td>
<td>Peso argentino</td>
</tr>
<tr>
<td>7</td>
<td>Peru</td>
<td>PE</td>
<td>7</td>
<td>Nuevo sol</td>
</tr>
<tr>
<td>8</td>
<td>Venezuela</td>
<td>VE</td>
<td>8</td>
<td>Bolivar fuerte</td>
</tr>
<tr>
<td>9</td>
<td>Colombia</td>
<td>CO</td>
<td>9</td>
<td>Peso colombiano</td>
</tr>
<tr>
<td>10</td>
<td>Ecuador</td>
<td>EC</td>
<td>10</td>
<td>US Dolar</td>
</tr>
<tr>
<td>11</td>
<td>Nicaragua</td>
<td>NI</td>
<td>11</td>
<td>Cordoba oro</td>
</tr>
</tbody>
</table>
Here are some CDR examples:

```xml
<cdrs>
  <cdr>
    <id_service_provider>DBProvider</id_service_provider>
    <id_application>PostgreSQL</id_application>
    <id_event>Subscription</id_event>
    <id_correlation>8405_1326350520_20130213T09:12:01</id_correlation>
    <purchase_code>123</purchase_code>
    <parent_app_id>FleetManagementApplication</parent_app_id>
    <product_class>PaaS</product_class>
    <description>Monthly Postgre Fee</description>
    <cost_currency>1</cost_currency>
    <cost_units>1300,0000</cost_units>
    <tax_currency>1</tax_currency>
    <tax_units>0,0000</tax_units>
    <cdr_source>1</cdr_source>
    <id_operator>1</id_operator>
    <id_country>1</id_country>
    <time_stamp>2013/02/13 09:12:01</time_stamp>
    <id_user>customer</id_user>
  </cdr>
  <cdr>
    <id_service_provider>ServerProvider</id_service_provider>
    <id_application>JOnAS-Orchestra</id_application>
  </cdr>
</cdrs>
```
<id_event>Subscription</id_event>
<id_correlation>8406_1521976590_20130213T09:12:06</id_correlation>
<purchase_code>124</purchase_code>
<parent_app_id>FleetManagementApplication</parent_app_id>
<product_class>PaaS</product_class>
<description>Monthly Jonas Fee</description>
<cost_currency>1</cost_currency>
<cost_units>780,000</cost_units>
<tax_currency>1</tax_currency>
<tax_units>0,000</tax_units>
<cdr_source>1</cdr_source>
<id_operator>1</id_operator>
<id_country>1</id_country>
<time_stamp>2013/02/13 09:12:06</time_stamp>
<id_user>customer</id_user>
</cdr>
<cdr>
<id_service_provider>ServerProvider</id_service_provider>
<id_application>ApacheServiceMix</id_application>
<id_event>Pay per use event</id_event>
<id_correlation>8407_1474276322_20130213T09:11:56</id_correlation>
<purchase_code>125</purchase_code>
<parent_app_id>FleetManagementApplication</parent_app_id>
<product_class>PaaS</product_class>
<description>ASM fee per call</description>
<cost_currency>1</cost_currency>
<cost_units>1,000</cost_units>
<tax_currency>1</tax_currency>
<tax_units>0,000</tax_units>
<cdr_source>1</cdr_source>
<id_operator>1</id_operator>
<id_country>1</id_country>
<time_stamp>2013/02/13 09:11:56</time_stamp>
<id_user>customer</id_user>
</cdr>
<cdr>
<id_service_provider>ServerProvider</id_service_provider>
<id_application>ApacheServiceMix</id_application>
<id_event>Pay per use time</id_event>
</id_event>
</id_correlation>8407_204511550_20130213T09:11:56</id_correlation>
<purchase_code>126</purchase_code>
</parent_app_id>FleetManagementApplication</parent_app_id>
</product_class>PaaS</product_class>
<description>ASM fee per tenant and hour</description>
<cost_currency>1</cost_currency>
<cost_units>0,1000</cost_units>
<tax_currency>1</tax_currency>
<tax_units>0,0000</tax_units>
<cdr_source>1</cdr_source>
</id_operator>1</id_operator>
</id_country>1</id_country>
<time_stamp>2013/02/13 09:11:56</time_stamp>
</id_user>customer</id_user>
</cdr>
</cdrs>

Refund example:

<cdrs>
<cdr>
<cdr_type>Refund</cdr_type>
<refund_reason>Cancelation</refund_reason>
</id_service_provider>DBProvider</id_service_provider>
</id_application>PostgreSQL</id_application>
</id_event>Subscription</id_event>
</id_correlation>8405_132632456_20130213T12:16:07</id_correlation>
<purchase_code>123</purchase_code>
</parent_app_id>FleetManagementApplication</parent_app_id>
</product_class>PaaS</product_class>
<description>Monthly Postgre Fee</description>
<cost_currency>1</cost_currency>
<cost_units>1300,0000</cost_units>
<tax_currency>1</tax_currency>
<tax_units>0,0000</tax_units>
<cdr_source>1</cdr_source>
</id_operator>1</id_operator>
</id_country>1</id_country>
<time_stamp>2013/02/13 09:12:01</time_stamp>
8.3.1.3 **Send a CDR with cURL**

You can also send a CDR using the following cURL command (just only one line):

```bash
curl -X POST -H 'Content-Type: application/xml' -d '"
  <id_user>customer</id_user>
  </cdr>
</cdrs>

8.3.1.4 **Raw request and response example**

This section shows the information of a real request and response to the RSS API.

REQUEST:

```
POST http://[hostname]:8080/fiware-rss/rss/cdrs/ HTTP/1.1
Accept-Encoding: gzip, deflate
Content-Type: application/xml
Content-Length: 13
Host: [hostname]:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

RESPONSE:

```
HTTP/1.1 200 OK
```
8.3.2 Balance and Accumulated API REST service

This API offers services by which a provider or/and a customer could establish expenditure limits for its customers/himself for a specific limit of time. Once exceeded this limit, it will not be possible to purchase anything until next period of time.

This API exposes the functionality that a partner will use to check the balance and accumulated expenses of a given customer, previously to accept a purchase.

The Balance and Accumulated API is located at:

http://host:port/expenditureLimit/balanceAccumulated

There is a way of obtaining the services offered by the API, using the following URL in a browser:

http://host:port/expenditureLimit/balanceAccumulated?_wadl

It shows the different services and operations.

The following sections explain the structure of the API and the information sent and received in each query.

8.3.2.1 Balance and accumulated API process

This process manages the balance and accumulated expenses of a customer.

Due to this process is an external process, it has to be done in two steps:

- Check balance: This operation allows the service provider to check if a customer has enough balance to perform a purchase without exceeding the expenses limit.
- Update accumulated expenses: Once the payment has been done, it is mandatory to update the accumulated expenses to be taken into account in the next operation.

8.3.2.1.1 Operations offered

This section explains the operations offered. In the next sections, take into account the following considerations:

- {apiRoot}: http://host:port/expenditureLimit/balanceAccumulated
- {userId}: Customer identifier

8.3.2.1.2 Check the balance of a customer

This operation allows checking if the customer has enough balance to purchase an application without exceeding the expenses limit. The descriptions of request and response are:

REQUEST:

- Resource:{apiRoot}/{userId}
- Operation: POST
- Content-Type: application/json
- Body: ExpendControlType entity
RESPONSE:
- HTTP Code: '200 OK'
- Content-Type: application/json
- Body: AccumsExpendType entity

An example of the use:

REQUEST:

```
POST /expenditureLimit/balanceAccumulated/userId01 HTTP/1.1

Host: host:port
Content-Type: application/json
Accept: application/json
Content-Length: nnnn

{  "service": "fiware",
   "appProvider": "comwet",
   "currency": "EUR",
   "chargeType": "C",
   "amount": 10
}
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn

{"service": "fiware",
 "appProvider": "Microsoft",
 "accums": [
  {"type": "daily",
   "currency": "EUR",
   "expensedAmount": 10
  },
  {"type": "weekly",
   "currency": "EUR",
   "expensedAmount": 50,
```
8.3.2.1.3 Update the cumulative expenses of a customer

This operation allows updating the cumulative expenses of a customer after performing a purchase. The descriptions of request and response are:

REQUEST:
- Resource: {apiRoot}/{userId}
- Operation: PUT
- Content-Type: application/json
- Body: ExpendControlType entity

RESPONSE:
- HTTP Code: ‘200 OK’
- Content-Type: application/json
- Body: AccumsExpendType entity

An example of the use:

REQUEST:
PUT/expenditureLimit/balanceAccumulated/userId01 HTTP/1.1

Host: host:port

Content-Type: application/json
Accept: application/json
Content-Length: nnnn

{  "service": "fiware",
   "appProvider": "comwet",
   "currency": "EUR",
   "chargeType": "C",
   "amount": 10
}

RESPONSE:
HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: nnnn

{"service": "fiware",
"appProvider": "Microsoft",
"accums": [
{"type": "daily",
"currency": "EUR",
"expensedAmount": 20
},
{"type": "weekly",
"currency": "EUR",
"expensedAmount": 60,
"nextPeriodStartDate": "2013-04-18T00:00:00.000Z"
},
{"type": "monthly",
"currency": "EUR",
"expensedAmount": 210,
"nextPeriodStartDate": "2013-05-01T00:00:00.000Z"
}]

8.3.2.1.4 Get the cumulative expenses of a customer

This operation allows getting the cumulative expenses of a customer. The descriptions of request and response are:

REQUEST:
- Resource: {apiRoot}/{userId}?QueryParameters
- Operation: GET
- Body: None

The query parameters are:

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>Service identifier</td>
</tr>
<tr>
<td>appProvider</td>
<td>Yes</td>
<td>Provider identifier</td>
</tr>
<tr>
<td>currency</td>
<td>Yes</td>
<td>Currency identifier (ISO CODE)</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Type of limit. If no type, all the limits are returned</td>
</tr>
</tbody>
</table>

**RESPONSE:**
- HTTP Code: '200 OK'
- Content-Type: application/json
- Body: AccumsExpendType entity

**Example of use:**

**REQUEST:**

```
GET
//expenditureLimit/balanceAccumulated/userId01?service=fiware&currency=EUR&appProvider=conwet HTTP/1.1

Host: host:port
```

**SUCCESSFUL RESPONSE:**

```
HTTP/1.1 200 OK

Content-Type: application/json
Content-Length: nnnn

{"service": "fiware",
 "appProvider": "conwet",
 "accums": [
     {
         "type": "daily",
         "currency": "EUR",
         "expensedAmount": 20
     },
     {
         "type": "weekly",
         "currency": "EUR",
         "expensedAmount": 60,
         "nextPeriodStartDate": "2013-04-18T00:00:00.000Z"
     },
     {
         "type": "monthly",
         "currency": "EUR",
         "expensedAmount": 210,
         "nextPeriodStartDate": "2013-05-01T00:00:00.000Z"
     }
 ]
```
8.3.2.1.5 Delete the cumulative expenses of a customer

This operation resets the cumulative expenses of a customer. The descriptions of request and response are:

REQUEST:
- Resource:{apiRoot}/{userId}/reset
- Operation: PUT
- Content-Type: application/json
- Body: None

RESPONSE:
- HTTP Code: ’200 OK’

An example of use:

REQUEST:

```
PUT /expenditureLimit/balanceAccumulated/userId01/reset HTTP/1.1
Host: host:port
Content-Type: application/json
{
    "service": "fiware",
    "appProvider": "conwet",
    "currency": "EUR"
}
```

SUCCESSFUL RESPONSE:
```
HTTP/1.1 200 OK
```

8.3.2.2 Complex types

This section contains the types used by requests and responses.

8.3.2.2.1 ExpendControlType

This structure contains the information of the request of check/update/reset expenditure limit.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>String</td>
<td>Service identifier.</td>
</tr>
<tr>
<td>appProvider</td>
<td>Yes</td>
<td>String</td>
<td>Provider/Store identifier.</td>
</tr>
<tr>
<td>currency</td>
<td>Yes</td>
<td>String</td>
<td>Currency (specified according ISO 4217 standard).</td>
</tr>
<tr>
<td>chargeType</td>
<td>No</td>
<td>String</td>
<td>Type of operation: Charge(C) or Refund(R). Required in check</td>
</tr>
</tbody>
</table>
and update operation.

| amount | No | Number | Amount of money to add to the accumulate. |

8.3.2.2.2 *AccumsExpendType*
This structure contains the information regarding the response of a request to the API.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>String</td>
<td>Service identifier.</td>
</tr>
<tr>
<td>appProvider</td>
<td>Yes</td>
<td>String</td>
<td>Provider/Store identifier.</td>
</tr>
<tr>
<td>accums[]</td>
<td>No</td>
<td>AccumExpendType</td>
<td>Accumulated expenses in different periods of times for a given customer.</td>
</tr>
</tbody>
</table>

8.3.2.2.3 *AccumExpendType*
This structure contains the information of an expenditure limit.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Yes</td>
<td>String</td>
<td>The type of limit can be expressed by:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;daily&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;weekly&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;monthly&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;perTransaction&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A &quot;perTransaction&quot; means that each individual transaction is treated in an individual way.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;daily&quot;, &quot;weekly&quot; and &quot;monthly&quot; types will be calculated during a specific period of time.</td>
</tr>
<tr>
<td>currency</td>
<td>Yes</td>
<td>String</td>
<td>Currency (specified according ISO 4217 standard).</td>
</tr>
<tr>
<td>nextPeriodStartDate</td>
<td>No</td>
<td>DateTime</td>
<td>Begining of the next accumulate period. This value is calculated by the system.</td>
</tr>
<tr>
<td>expensedAmount</td>
<td>Yes</td>
<td>Number</td>
<td>Accumulated at the current time (Not allowed for: requests nor in &quot;perTransaction&quot; type)</td>
</tr>
</tbody>
</table>

8.3.2.3 *Fault Exceptions*
Each time an exception occurs inside the system, a structure containing information regarding to the exception happened is returned. Apart from that, a HTTP error code will be returned in the HTTP header (500,404,400,403).

The information structure of an exception is:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Yes</td>
<td>String</td>
<td>Code (exceptionId) of the type of exception.</td>
</tr>
</tbody>
</table>
An example of exception is:

```json
{
    "code": "SVC0001",
    "description": "Generic Client Error"
}
```

Some of the common exception codes are:
- SVC1000: Missing Mandatory parameter.
- SVC1001: Invalid parameter.
- SVC0002: Invalid parameter value.
- SVC1006: Resource does not exist.
- SVC3705: Insufficient payment method balance.
- SEC1004: Invalid Token.
- GRT0001: Generic Exception.

8.3.3 Expenditure limit management API REST service

This API offers services by which a provider or/and a customer could establish expenditure limits for its customers/himself for a specific limit of time. Once exceeded this limit, it will not be possible to purchase anything until next period of time.

This API exposes the functionality that a partner will use to manage (create, modify or delete) the expenditure limits that will be taken into account.

The expenditure limits management API is located at:

```
http://host:port/expenditureLimit/limitManagement
```

There is a way of obtaining the services offered by the API, using the following URL in a browser:

```
http://host:port/expenditureLimit/limitManagement?_wadl
```

It shows the different services and operations. Two types of expenditure limits can be established: one from provider's point of view and other from customer's point of view.

The following sections explain the structure of the API and the information sent and received in each petition.

8.3.3.1 Wildcard Values

In order to be able to define limits that apply to more than one user, wildcard values will be introduced. The limits will be taken into account from a more restrictive point of view to the less. It is possible to use to types of wildcards:
- “noUserId”: This wildcard is used to specify that a limit applies to every user. This is used for provider limits purpose.
- “noProviderId”: This wildcard is used to specify that a limit applies to every provider. This is used to specify default limits purpose.

8.3.3.2 Operations offered

This section explains the operations offered.
In the next sections, take into account the following considerations:

- \{apiRoot\}: \texttt{http://host:port/expenditureLimit/limitManagement}
- \{userId\}: Customer identifier
- \{appProviderId\}: Provider Identifier

### 8.3.3.2.1 Create/update provider expenditure limit information

This operation allows creating or updating the general data related to the expenditure limit for any user in a given application provider. The update (modification) operation is considered as a ‘delete and creation’ operation because it will replace the previous data. The descriptions of request and response are:

**REQUEST:**

- Resource:\{apiRoot\}/\{appProviderId\}
- Operation: POST
- Content-Type: application/json
- Body: LimitGroupType entity

**RESPONSE:**

- HTTP Code: ‘200 OK’
- Content-Type: application/json
- Body: LimitGroupType entity

An example of the use:

**REQUEST:**

```
POST /expenditureLimit/limitManagement/appproviderId01 HTTP/1.1
Host: host:port
Content-Type: application/json
Accept: application/json

{ "service":"fiware",
  "limits": [
    { "type": "perTransaction",
      "currency": "EUR",
      "maxAmount": 1000
    },
    { "type": "weekly",
      "currency": "GBP",
      "maxAmount": 500,
      "notificationAmounts": [400]
  ]
}
```

---

**D.3.4.3: FI-WARE User and Programmer Guide**
RESPONSE:

HTTP/1.1 201 Created

Content-Type: application/json

Content-Length: nnnn

{
    "service": "fiware",
    "limits": [
        {
            "type": "perTransaction",
            "currency": "EUR",
            "maxAmount": 1000
        },
        {
            "type": "weekly",
            "currency": "GBP",
            "maxAmount": 500,
            "notificationAmounts": [400]
        },
        {
            "type": "daily",
            "currency": "EUR",
            "maxAmount": 200,
            "notificationAmounts": [150, 170, 190]
        },
        {
            "type": "monthly",
            "currency": "EUR",
            "maxAmount": 5000,
            "notificationAmounts": [3000, 4000, 4500]
        },
        {
            "type": "monthly",
            "currency": "GBP",
            "maxAmount": 3000,
            "notificationAmounts": [2000, 2500]
        }
    ]
}
8.3.3.2.2 Query for provider expenditure limit information

This operation allows getting the limits information of a provider. The descriptions of request and response are:

REQUEST:
- Resource: {apiRoot}/providerId?QueryParameters
- Operation: GET
- Body: None

The query parameters are:

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>Service identifier</td>
</tr>
<tr>
<td>currency</td>
<td>No</td>
<td>Currency identifier (ISO CODE)</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Type of limit. If no type, all the limits are returned</td>
</tr>
</tbody>
</table>

RESPONSE:
- HTTP Code: '200 OK'
- Content-Type: application/json
- Body: LimitGroupType entity

Example of use:

REQUEST:

GET //expenditureLimit/limitManagement/appProviderId01?service=fiware
HTTP/1.1
Host: host:port

SUCCESSFUL RESPONSE:

HTTP/1.1 200 OK

Content-Type: application/json
Content-Length: nnnn

{ "service":"fiware",
  "limits": [ 
    {"type": "perTransaction",
     "currency": "EUR",
     "maxAmount": 10 },
    { "type": "daily",
     "currency": "EUR",
     "maxAmount": 20,
     "notificationAmounts": [15] },
    { "type": "monthly",
     "currency": "EUR",
     "maxAmount": 500,
     "notificationAmounts": [400, 450] }
  ]
}

8.3.3.2.3  Delete expenditure limit provider information

This operation allows deleting the user expenditure limit information and implies the cascade deletion of the pairs appProvider-user where the application provider is appProviderId.

REQUEST:

- Resource:{apiRoot}/{appProviderId}?QueryParameters
- Operation: DELETE
- Content-Type: application/json
- Body: None

The query parameters are:

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>Service identifier</td>
</tr>
<tr>
<td>currency</td>
<td>No</td>
<td>Currency identifier (ISO CODE)</td>
</tr>
<tr>
<td>----------</td>
<td>----</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Type of limit. If no type, all the limits are returned</td>
</tr>
</tbody>
</table>

RESPONSE:
- HTTP Code: '200 OK'

An example of use:
REQUEST:
```
DELETE /expenditureLimit/limitManagement/appProviderId01?service=fiware HTTP/1.1
```

SUCCESSFUL RESPONSE:
```
HTTP/1.1 200 OK
```

8.3.3.2.4  *Create/update user expenditure limit information*

This operation allows the creation or update of the expenditure limit data related to a specific user and a specific application provider. The update (modification) operation is considered as a 'delete and creation' operation because it will replace all the previous data. The descriptions of request and response are:

REQUEST:
- Resource:{apiRoot}/{appProviderId}/{userId}
- Operation: POST
- Content-Type: application/json
- Body: LimitGroupType entity

RESPONSE:
- HTTP Code: '200 OK'
- Content-Type: application/json
- Body: UserExpenditureLimitInfoType entity

An example of the use:
REQUEST:
```
POST /expenditureLimit/limitManagement/appProviderId01/userId01 HTTP/1.1
Host: host:port
Content-Type: application/json
Accept: application/json
Content-Length: nnnn
{  "service":"fiware",
```
"limits": [
  {
    "type": "perTransaction",
    "currency": "EUR",
    "maxAmount": 10
  },
  {
    "type": "daily",
    "currency": "EUR",
    "maxAmount": 20
  },
  {
    "type": "monthly",
    "currency": "EUR",
    "maxAmount": 500,
    "notificationAmounts": [400, 450]
  }
],

"appProvidersLimits": [
  {
    "type": "perTransaction",
    "currency": "EUR",
    "maxAmount": 100
  },
  {
    "type": "monthly",
    "currency": "EUR",
    "maxAmount": 500,
    "notificationAmounts": [300, 400, 450]
  }
]

RESPONSE:

HTTP/1.1 201 Created

Content-Type: application/json

Content-Length: nnnn

{  "service": "fiware",
  "appProviderId": "appProviderId01",
  "userLimits": [
    {
      "type": "perTransaction",
      "currency": "EUR",
      "maxAmount": 100
    },
    {
      "type": "monthly",
      "currency": "EUR",
      "maxAmount": 500,
      "notificationAmounts": [300, 400, 450]
    }
  ],
  "appProvidersLimits": [
    {
      "type": "perTransaction",
      "currency": "EUR",
      "maxAmount": 100
    },
    {
      "type": "monthly",
      "currency": "EUR",
      "maxAmount": 500,
      "notificationAmounts": [300, 400, 450]
    }
  ]}
8.3.3.2.5 Query for user expenditure limit information

This operation allows getting the expenditure limits information of a customer. The descriptions of request and response are:

REQUEST:
- Resource:{apiRoot}/{appProviderId}/{userId}?QueryParameters
- Operation: GET
- Body: None

The query parameters are:

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>Service identifier</td>
</tr>
<tr>
<td>currency</td>
<td>No</td>
<td>Currency identifier (ISO CODE)</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Type of limit. If no type, all the limits are returned</td>
</tr>
</tbody>
</table>

RESPONSE:
- HTTP Code: ‘200 OK’
- Content-Type: application/json
• Body: LimitGroupType entity

Example of use:

REQUEST:

```plaintext
GET
//expenditureLimit/limitManagement/appProviderId01/userId01?service=fiware
HTTP/1.1
Host: host:port
```

SUCCESSFUL RESPONSE:

```
HTTP/1.1 200 OK

Content-Type: application/json
Content-Length: nnnn

{
  "service": "fiware",
  "appProviderId": "appProviderId01",
  "userLimits": [
    {
      "type": "perTransaction",
      "currency": "EUR",
      "maxAmount": 100
    },
    {
      "type": "monthly",
      "currency": "EUR",
      "maxAmount": 500,
      "notificationAmounts": [300, 400, 450]
    }
  ],
  "appProvidersLimits": [
    {
      "type": "perTransaction",
      "currency": "EUR",
      "maxAmount": 100
    },
    {
      "type": "monthly",
      "currency": "EUR",
      "maxAmount": 500,
      "notificationAmounts": [300, 400, 450]
    }
  ]
}```
"serviceLimits": [
  {
    "type": "perTransaction",
    "currency": "EUR",
    "maxAmount": 100
  },
  {
    "type": "monthly",
    "currency": "EUR",
    "maxAmount": 1000,
    "notificationAmounts": [700, 900, 950]
  }
]
}

8.3.3.2.6 **Delete expenditure limit user information**

This operation delete the expenditure limit data related to a specific user and a specific application provider:

**REQUEST:**
- Resource: `{apiRoot}/{appProviderId}/{appProviderId}/{userId}?QueryParameters`
- Operation: DELETE
- Content-Type: application/json
- Body: None

The query parameters are:

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>Service identifier</td>
</tr>
<tr>
<td>currency</td>
<td>No</td>
<td>Currency identifier (ISO CODE)</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Type of limit. If no type, all the limits are returned</td>
</tr>
</tbody>
</table>

**RESPONSE:**
- HTTP Code: ‘200 OK’

An example of use:

**REQUEST:**
```
DELETE /expenditureLimit/limitManagement/appProviderId01/userId01?service=fiware
HTTP/1.1
```

**SUCCESSFUL RESPONSE:**
```
HTTP/1.1 200 OK
```
8.3.3.3 **Complex types**
This section contains the types used by requests and responses.

8.3.3.3.1 **LimitGroupType**
This structure combines the data for the expenditure limits to apply over the user payments in a global way or by default.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>String</td>
<td>Service for which the limits are defined.</td>
</tr>
<tr>
<td>limits</td>
<td>No</td>
<td>LimitType[0..N]</td>
<td>User spending limits for the sum of all users' application providers charges.</td>
</tr>
</tbody>
</table>

8.3.3.3.2 **LimitType**
This structure contains the information of an expenditure limit. This structure contains the information of the request of check/update/reset expenditure limit.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Yes</td>
<td>String</td>
<td>The type of limit can be expressed by:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;daily&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;weekly&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;monthly&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;perTransaction&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A &quot;perTransaction&quot; means that each individual transaction is treated in an individual way.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;daily&quot;, &quot;weekly&quot; and &quot;monthly&quot; types will be calculated during a specific period of time.</td>
</tr>
<tr>
<td>currency</td>
<td>Yes</td>
<td>String</td>
<td>Currency (specified according ISO 4217 standard).</td>
</tr>
<tr>
<td>maxAmount</td>
<td>No</td>
<td>Number</td>
<td>Maximal amount for the user to expend. No limit is applied if negative or no value is given.</td>
</tr>
<tr>
<td>notificationsAmounts</td>
<td>No</td>
<td>Number[0..N]</td>
<td>Amounts for which a notification will be sent if they are reached by the expenses.</td>
</tr>
</tbody>
</table>

8.3.3.3.3 **UserExpenditureLimitInfoType**
This structure combines all the information related to the expenditure limit for a user.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Yes</td>
<td>String</td>
<td>Service for which the limits are defined.</td>
</tr>
<tr>
<td>userLimits</td>
<td>Yes</td>
<td>LimitType[0..N]</td>
<td>Information for the general expenditure limits to apply over the user payments.</td>
</tr>
</tbody>
</table>
### 8.3.3.4 Fault Exceptions

Each time an exception occurs inside the system, a structure containing information regarding to the exception happened is returned. Apart from that, a HTTP error code will be returned in the HTTP header (500, 404, 400, 403).

The information structure of an exception is:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Yes</td>
<td>String</td>
<td>Code (exceptionId) of the type of exception.</td>
</tr>
<tr>
<td>description</td>
<td>Yes</td>
<td>String</td>
<td>Description of the specific exception.</td>
</tr>
</tbody>
</table>

An example of exception is:

```json
{
  "code": "SVC0001",
  "description": "Generic Client Error"
}
```

Some of the common exception codes are:

- SVC1000: Missing Mandatory parameter.
- SVC1001: Invalid parameter.
- SVC0002: Invalid parameter value.
- SVC1006: Resource does not exist.
- SEC1004: Invalid Token.
- GRT0001: Generic Exception.
9 Store - W-Store - User and Programmer Guide

9.1 Introduction
This page contains the user and programmer guide for WStore, a reference implementation of the Store Generic Enabler.

9.2 User Guide
This user guide contains a description of the different tasks that can be performed in WStore using its web interface from different points of view, depending on the roles of the corresponding user.

9.2.1 Profile configuration
All users can configure their profile including some information such as a default tax address.
To configure the user profile, move your mouse to username on the top right of the page and choose settings.

In the displayed modal, it is possible to view user profile info. To change this info or provide new one, select the Edit button.
Fill the different fields through the Tabs and press *Update*. 
9.2.2 Service provider

This section explains how a service provider can create and monetize her service offerings using WStore GUI.

9.2.2.1 Registering a resource

Resources are used in the offerings in order to allow to include downloadable digital assets, such as applications, widgets, etc.

To register a resource, select the Catalogue view.
Select the options dropdown and choose *Register resource*
In the displayed modal, the first step is filling the resource name, the resource version and the resource description. Then, it is necessary to provide a mime type for the resource. Finally, it is necessary to upload the resource or provide a link from where the resource can be downloaded.
9.2.2.2 Viewing resources

A service provider is able to view the resources which are already registered. To view the resources, select the catalogue view and choose View resources in the dropdown menu.
In the displayed modal, you can see the different resources you have registered.
9.2.2.3  Creating an offering

Offerings are the main entity managed by WStore and include all the relevant information such as the pricing model, legal conditions, interactions, service level agreement, etc. To create an offering, select the Catalogue view and choose Create offering from dropdown menu.
In the displayed modal, fill the name, version and description fields. Next, provide an image and an optional set of screenshots. Finally, select how to provide the notification URL. This field is used by WStore to notify the service provider when its offering has been purchased. There are three different options: (a) Provide a new notification URL for this offering. (b) Use the default notification URL of the provider that can be configured in the user profile configuration form. (c) Not using a notification URL for this offering.
The next step is providing an USDL document describing the offering. There are three possible options: (a) Create a simple USDL just by using the provided form. (b) Upload a USDL document. (c) Provide an URL pointing to the USDL document if it as been previously uploaded into a repository.
Once the USDL info has been provided, the next step consist of including Applications. This applications need to be understood as OAuth2 Applications and are those registered in the Identity
Manger by the provider. Including Applications in an offering allows to grant real access to the related services via OAuth2 to the customers that acquire the offering.

The final step consist of selecting resources previously registered by the provider.
9.2.2.4 Managing an offering

Updating an offering

It is possible to update an offering info by providing a new USDL description or including new images.

To update an offering, select the catalogue view and the *Provided* tab. This tab contains the offerings provided by the service provider.
Select the offering to be updated. Note that only offerings with uploaded state (Offerings that have not been published yet) can be updated.
In the advanced operations, select *Edit*.
In the displayed modal, it is possible to provide a USDL as in the offering creation. It is also possible to provide some screenshots or a new logo.
Binding resources
Once an offering has been created it is still possible to manage the included resources. To bind resources, select the Catalogue view and the Provided Tab. Then select the offering to be bound. Note that only offerings with uploaded state can be updated.

Select the “Bind resources” option.
In the displayed modal, select the resources to be bound and press Accept
Note that this operation is an absolute update, that is, the selected resources are the bound resources. Therefore, it is possible to bind and unbind resources in the same action.

**Publishing an offering**

Publishing an offering means start selling it. To publish an offering select the catalogue view and the *Provided* Tab. Then select the offering to be published. In the offering details view select the *Publish* option.
In the displayed modal, select the Marketplaces where publish the offerings. Note that selecting a Marketplace is not mandatory.

The offering is now **Published** and cannot be updated.
Tagging an offering

It is possible for an offering provider to tag their offerings. To tag an offering, select the Catalogue view and the Provided Tab. Then select the offering to be tagged.

Select the Update tags option

Include the different tags and press Accept
You may also be suggested some tags that may fit your offering.

Deleting an offering

The action of deleting an offering has different effects depending on its state. If the offering has not been published it is completely deleted from WStore. However, if the offering has been published, its state changes to deleted and cannot be purchased anymore, but customers that have already purchased it, still has access to the offering and its resources.

To delete an offering, select the Catalogue view and the Provided Tab. Then select the offering to be deleted.

In the offering details view, select the Delete offering option.
Select accept in the displayed window.

If the offering has been published the option *Delete* replaces *Publish* as main action.
9.2.3 Customer

This section explains how a customer can search and acquire offerings using WStore GUI.

9.2.3.1 Searching for offerings

There are some options for searching offerings in WStore. As it can be seen in the following image, the main page contains the Top rated and the newest offerings.
To search using a keyword type it in the textbox and press **Search**.
The offerings that match the search are shown.
It is also possible to view all the offerings selecting the View All button.

9.2.3.2 Purchasing an offering

The first step to purchase a published offering is selecting it after searching. To start with the purchasing process click on the button included in the offering.
Alternatively, it is possible to select the *Purchase* button in the offering details view.
In the displayed modal, the first step is providing a tax address for the purchase. It is possible to use the default tax address from the user profile by clicking the checkbox *Use user profile tax address*. Then, select *Accept*. 
In case the offering can be acquired under different pricing models, the first step is selecting the plan.
WStore informs that the payment process will continue in a separate window.

WStore redirects the browser to the PayPal confirmation page.

Introduce your PayPal credentials and confirm the payment.
Return to WStore page and end the process by closing the displayed window.
9.2.3.3 **Downloading resources and invoices**

To download the resources and the invoices of a purchased offering, select the *Catalogue* view and the *Purchased* Tab. Then, select the offering.

Select the *Download* button.
In the displayed modal, is possible to download invoices and resources by clicking on the link.
9.2.3.4 **Commenting an offering**

To comment and rate an offering, select a purchased offering and click on the *Comment* button situated in the comments section.

Fill the number of stars and give a comment.
9.2.4 Admin

This section describes the different tasks that can be performed by an admin user using WStore GUI.

9.2.4.1 Registering WStore on a Marketplace

WStore can be registered on a Marketplace in order to allow service providers to publish their offerings on them, making their offerings available to potential customers that search for offerings in the Marketplace.

Note that this process is made from WStore GUI, since WStore needs to have information about in what Marketplaces is registered on.

To register WStore on a Marketplace, select the Administration view.
Press the Add symbol of the *Marketplaces* row.
Fill the internal name and the host of the Marketplace.

Pressing on the *Marketplaces* row is possible to view in what Marketplaces WStore is registered on.
9.2.4.2 **Registering a Repository on WStore**

It is possible to register some instances of the Repository GE in order to allow service providers to Upload USDL documents directly when creating an offering.

To register a Repository on WStore select the *Administration* view and press the Add symbol of the *Repositories* row.

Fill the internal name and the host of the Repository.
Pressing on the *Repositories* row is possible to view what Repositories are registered on WStore.
9.2.4.3 **Registering a RSS on WStore**

It is possible to register RSS instances on WStore in order to perform the revenue sharing of the purchased offerings.

To register a RSS on WStore select the *Administration* view and press the Add symbol of the *RSS* row.

Fill the internal name and the host of the RSS, as well as the default expenditure limits for WStore.
Pressing on the RSS row is possible to view what RSSs are registered on WStore.
9.2.4.4 **Registering a Price Unit**

Price Units are used in order to determine the concrete pricing model that applies to an offering.

To include a new supported price unit select the *Administration* view and click the add symbol in the *Pricing model units* row.
Fill the name and the defined model of the unit. If the defined model is *Subscription* it is also necessary to specify the renovation period.
It is possible to view existing units by click on the *Pricing model units* row.

9.3 Programmer Guide

The programmer guide contains a description of the actions that can be performed by a developer, in order to integrate WStore capabilities with her solution using WStore API. For a complete description of WStore API view **Store GE Open API Specification**.
9.3.1 API Authentication and authorization

WStore API requires users to be authenticated and requires them to authorize developer’s application in order to access WStore API in their name. To perform this process WStore uses an OAuth2 approach.

Depending on the authorization mode of the WStore instance there are two possible ways for API authorization. If the WStore instance is using an idM GE, the developer application should include a valid token obtained from the idM in all the related requests. For information on how to authorize an application using the idM GE have a look at Identity Management GE User and Programmers Guide.

In case the WStore instance uses its own authentication mechanism the developer should follow the following process.

The first step consists on user authentication and application authorization. Note that the application should be registered in WStore in order to have a client_id and a client_secret. To perform this step is necessary to make the following request:

```
GET /oauth2/auth HTTP/1.1
Accept: application/json
```

This request must include the following params.

- **client_id**: Id of the application in WStore.
- **redirect_uri**: URI where WStore redirects when the call finishes
- **response_type**:

When this request is performed the user is redirected to a page where the user can log in.
and authorize the application.

Once the user has authorized the application, an authorization code is returned to the redirect_uri provided.

The next step is to acquire the access token. To perform this step, it is necessary to make the following request:
This request must include the following params.
- client_id: Id of the application in WStore
- client_secret: Secret of the application in WStore
- grant_type:
  - code: Authorization code provided in the previous step
  - redirect_uri: URI where WStore redirects when the call finishes

WStore responds to this request providing an access and a refresh token. The access token must be included as a header in all API requests, and the refresh token is used to acquire a new access token in case it expires.

To refresh the access token is necessary to make the following request:

```
POST /oauth2/token HTTP/1.1
```

This request must include the following params.
- client_id: Id of the application in WStore,
- client_secret: Secret of the application in WStore,
- grant_type:
  - refresh_token: refresh token provided in the previous step

9.3.2 Resources management integration

It is possible for a developer to integrate the Resources API in order to monetize different catalogues included in the developer solution. To perform this monetization, it is necessary to register the resources using a POST request, making them available to be bound in an offering.

9.3.2.1 Registering resources

The contents of the request depends on the resource characteristics and the developer criteria.

**Downloadable resource**

If the resource is a downloadable resource and the resource is provided, it is possible to provide the resource itself by creating a multipart request or encode it in base64 and include this encoding in the JSON.

```
POST /api/offering/resources HTTP 1.1
Content-type: multipart/form-data

{
  "name": "Smart City Lights Mashup",
  "version": "1.0",
  "description": "This resource contains a mashup for Smart City Lights",
  "content_type": "application/x-mashup+mashable-application-component"
}
```
Downloadable resource providing link

If the resource is a downloadable resource but the service provider has her own server to serve media files, s/he can provide an URL where the resource can be downloaded instead of the resource itself, making the request as follows.

```json
POST /api/offering/resources HTTP 1.1
Content-type: application/json
{
    "name": "Smart City Lights Mashup",
    "version": "1.0",
    "description": "This resource contains a mashup for Smart City Lights",
    "content_type": "application/x-mashup+mashable-application-component",
    "link": "https://downloadmashuplink.com/smartcity"
}
```

All this requests return a 201 code if everything is successful.
9.3.2.2  Getting resources

It is also possible to retrieve the information of the different resources belonging to the user making the following call.

```
GET /api/offering/resources HTTP 1.1
Accept: application/json
```

This call returns a list with the following format:

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Cookie

{
  [
    {
      "content_type": "application/x-mashup+mashable-application-component"
      "description": "Smart City Lights is an app"
      "name": "Smart City Management"
      "version": "1.0"
    }
  ]
}
```

9.3.3  Offerings management integration

WStore also offers the different operations to manage offerings through its API in order to allow developers to create different applications capable of performing this management and allowing external applications to enrich their resources with pricing models, service level agreements, etc.

9.3.3.1  Getting offerings

The next request shows how is possible to retrieve offerings using WStore API. The next call is supposed to return all the offerings published (its state is published) in WStore.

```
GET /api/offering/offers HTTP 1.1
Accept: application/json
```

WStore responds with a list of offerings with the following format:

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Cookie
```
[{
  "name": "SmartCityLights",
  "owner_organization": "CoNWeT",
  "owner_admin_user": "app_provider",
  "version": "1.0",
  "state": "published",
  "description_url": "http://examplerepository.com/storeCollection/SmartCityLights",
  "marketplaces": [example_marketplace],
  "resources": [{
    "name": "Smart City Lights Mashup",
    "version": "1.0",
    "description": "This resource contains a mashup for Smart City Lights",
  }],
  "applications": [{{
    "id": 18,
    "name": "Context broker",
    "url": "https://orion.lab.fi-ware.eu",
    "description": "Context broker"
  }],
  "rating": "5",
  "comments": [{{
    "date": "2013/01/16",
    "user": "admin",
    "rating": "5",
    "comments": "Good offering"
  }],
  "tags": [smart, city],
  "image_url": "http://examplestore.com/media/image",
  "related_images": [],
  "offering_description": {parsed USDL description info}
}]

The applications field only appears if an identity manager is being used. This field contains the different OAuth2 applications offered in the offering.
Note that the resources field contains the information of the resources bound to the offering. In case that the offering had been purchased by the user making the call and that the resource was a downloadable resource, this field contains also a URL where to download the resource. As it is mentioned above the previous call returns all offerings whose state is published. However it is possible to configure the API call using query strings in order to limit the number, select the first offering expected or ask for the user offerings (provided and purchased). To perform this calls the following query strings can be used and combined.

- filter=published: Returns published offerings
- filter=purchased: Returns the offerings purchased by the user making the call
- filter=provided: Returns the offerings provided by the user making the call

- sort=date: Sorts the returned offerings using the date (default).
- sort=name: Sorts the returned offerings using the name.
- sort=popularity: Sorts the returned offerings using the popularity.

- start=<number>: Defines the first offering to be returned
- limit=<number>: Defines the number of offerings to be returned

This two query strings can be combined with the filter query string and are used to paginate the results. Additionally it is possible to use action=count. This query string can be combined with filter, and modifies the functionality of the call that now returns the number of offerings instead of the offerings info. It is also possible to request a single offering by making the following request:

```
GET /api/offering/offerings/{organization}/{name}/{version} HTTP 1.1
Accept: application/json
```

The response of that call is similar as the previous one but only an offering is returned instead of a list of offerings.

9.3.3.2 Searching offerings

WStore allows to search for offerings using a keyword. To perform this action it is necessary to make the following call:

```
GET /api/search/{keyword} HTTP 1.1
Accept: application/json
```

In this case the response is similar to the getting offerings call but only offerings that satisfy the keyword are returned.

9.3.3.3 Creating an offering

WStore allows to create new offerings through its offering API. There are three different ways of creating an offering depending on the state of the USDL describing the offering. If the user has the USDL, s/he can include it in the request as in the following call:

```
POST /api/offering/offerings HTTP 1.1
Content-Type: application/json
{  
```
If the USDL is uploaded previously into a repository then the user can include its URL as in the following request.

```json
POST /api/offering/offerings HTTP 1.1
Content-Type: application/json
{
    "name": "example_offering",
    "version": "1.0",
    "image": {
        "name": "catalogue.png",
        "data": <encoded_data>,
    },
    "applications": [{
        "id": 18,
        "name": "Context broker",
        "url": "https://orion.lab.fi-ware.eu",
        "description": "Context broker"
    }],
    "related_images": [],
    "repository": "testbed_repository",
    "resources": [{
        "provider": "app_provider",
        "name": "Smart City Management",
        "version": "1.0"
    }],
    "offering_description": {
        "content_type": "text/turtle",
        "data": "raw USDL document (RDF XML, N3, Turtle)"
    }
}
```
Finally, if the user does not have an USDL s/he can provide basic info that is used to create an USDL document in WStore.

```json
POST /api/offering/offerings HTTP 1.1
Content-Type: application/json
{
    "name": "example_offering",
    "version": "1.0",
    "image": {
        "name": "catalogue.png",
        "data": <encoded_data>,
    },
    "related_images": [],
    "resources": [{
        "provider": "app_provider",
        "name": "Smart City Management",
        "version": "1.0"
    }],
    "offering_info": {
        "description": "Description of the offering",
        "pricing": {
            "price_model": "free"
        },
        "legal": {
```
The images passed in this call are included in the JSON document directly, encoded in base64 format; moreover, screenshots must be included in the related_images field as a list of elements with the same format as the image field.

WStore responds to this call with a 201 Created code if the request is successful. Note that to perform this action the user making the call must have the provider role.

### 9.3.3.4 Updating an offering

WStore supports updating created offerings that have not been published yet. To update an uploaded offering, new logo, screenshots or USDL can be provided. Note that the name and version of the offering cannot be changed since are used to identify the offering. Similarly to the offering creation, there are three different ways of updating an offering depending on where the USDL is. If the USDL document is directly provided, this document overrides the previous USDL description in the repository; therefore, this method for updating can be used even if the offering was created using a repository link.

#### PUT /api/offering/offerings/{organization}/{name}/{version} HTTP 1.1

```
Content-Type: application/json

{
    "image": {
        "name": "catalogue.png",
        "data": <encoded_data>,
    },
    "related_images": [],
    "repository": "example_repository",
    "offering_description": {
        "content_type": "text/turtle",
        "data": "raw USDL document (RDF XML, N3, Turtle)"
    }
}
```

If a USDL link is provided, this URL must be the same as the provided when the offering was created. This method is the one used in case that multiple applications were writing over the USDL description.

#### PUT /api/offering/offerings/{organization}/{name}/{version} HTTP 1.1

```
Content-Type: application/json

{
    "image": {
```
If the information is included, the new created USDL will override the existing one.

WStore returns a 200 OK code in case the request is successful. Note that to perform this action, the user making the call must be the owner of the offering.

9.3.3.5 Binding resources

WStore supports bind registered resources with created offerings using the offering management API. To perform the binding, it is necessary to have the resources info, so may be useful to make a getting resources request as defined in the resources management section.
“resources”: [{
    "provider": "app_provider",
    "name": "Smart City Management",
    "version": "1.0"
  },
  
  {
    "provider": "app_provider",
    "name": "HistoryMod",
    "version": "1.0"
  }
}

WStore returns a 200 OK code in case the request is successful.

The binding process is an absolute update, that is, when the request finishes, the offering resources are the same as the contained in the request. Note that this action only can be performed if the offering is not published.

9.3.3.6 Publishing an offering

WStore supports publishing an offering using the offerings management API. Publishing an offering means start selling it. Note that in this request is possible to select in what Marketplaces (WStore must be registered on them) the offering is going to be published. It is also possible not publishing the offering in any Marketplace.

POST /api/offering/offerings/{organization}/{name}/{version}/publish

Content-Type: application/json

{

  “marketplaces”: ["testbed_marketplace",]
}

WStore returns a 200 OK code in case the request is successful.

The marketplaces list contains the name of the different Marketplaces as was included in WStore when WStore was registered in those marketplaces.

9.3.3.7 Deleting an offering

WStore supports to delete offerings via API. Note if the offering has been published it is not deleted but its state is changed to deleted.
DELETE /api/offering/offerings/{organization}/{name}/{version} HTTP 1.1

WStore returns a 204 No content code if the request is successful.

9.3.4 Purchases

9.3.4.1 Purchase API integration

WStore supports to integrate purchases with different external applications using the purchases API directly. Using this method to integrate purchases requires the developer to take into account the payment method since it is possible that it needs to redirect users to the PayPal confirmation page.

The requests to perform a purchase directly using the purchases API are different depending on the payment method selected.

Credit card

The following request shows how to perform a purchase using a credit card. If no tax address or credit card provided, then default values stored in user’s profile are used. Moreover, the field offering used to identify the offering to be purchased could contain different values, apart for the method used in the request (organization, name, version), it is also possible to provide the URL of the USDL in the Repository GEi (description_url field in offering requests), this method is useful to purchase offerings that have been searched in a Marketplace GEi. The plan label field is used to identify the price plan when there are more than one, if only a plan exists this field is not mandatory.

```
POST /api/contracting HTTP 1.1
Content-Type: application/json

{
    "offering": {
        "organization": "CoNWeT",
        "name": "SmartCityLights",
        "version": "1.0"
    },
    "plan_label": "update",
    "tax_address": {
        "street": "C/Los alamos n 17",
        "city": "Santander",
        "postal": "39011",
        "country": "Spain"
    }
    "payment_info": {
        "payment_method": "credit card",
        "credit_card": {
            "number": "546798367265",
            "type": "MasterCard",
            "expire_year": "2018",
```
WStore responds with a 201 Created code is the request is successful.

**PayPal**

The following request shows how to perform a purchase using a PayPal account. Note that if no tax address provided the default value is used.

```json
POST /api/contracting HTTP 1.1
Content-Type: application/json
Accept: application/json

{
  "offering": {
    "organization": "CoNWeT",
    "name": "SmartCityLights",
    "version": "1.0"
  },
  "plan_label": "update",
  "tax_address": {
    "street": "C/Los alamos n 17",
    "city": "Santander",
    "postal": "39011",
    "country": "Spain"
  },
  "payment_info": {
    "payment_method": "paypal"
  }
}
```

If the request is success WStore will respond with a redirection URL. This URL is created by PayPal and the user browser should be redirected to that window, since, PayPal requires user authentication and confirmation to perform the payment.

**Response:**

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Cookie
```
9.3.4.2 **Purchase redirection integration**

WStore also supports to integrate external applications with the purchase process using WStore web interface to perform the payment. To integrate an application using this method, the client application requests for a purchase formulary for a concrete offering and WStore responds with a redirection URL where the client application should redirect the user browser in order to start the purchasing process.

```json
{
    "redirection_link": "http://paypalredirectionlink.com/
}
```

POST /api/contracting/form HTTP 1.1
Content-Type: application/json
Accept: application/json

```json
{
    "offering": {
        "organization": "CoNWeT",
        "name": "SmartCityLights",
        "version": 1.0,
    },
    "redirect_uri": "http://customerredirecturi.com"
}
```

Note that the **offering** field, used to identify the offering, could also contain the URL pointing to the USDL description in the Repository GEi (**description_url** field in offering request) in order to allow to integrate WStore with a solution that uses a Marketplace GEi for searching offerings.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Cookie
```

```json
{
    "url": "http://wstore.lab.fi-ware.eu/contracting/form?ID=63865adf6c2ca6f7"
}
```

The URL returned should be used to redirect the user browser. This URL points to a formulary that allows the user to pay using the WStore GUI.

When the user ends the purchase, the window is closed and WStore sends a notification to the client application using the redirect URI provided in the call.
9.3.4.3 **Purchases notifications**

When a service provider publish an offering in WStore, s/he should provide an URL where s/he can receive a notification when her offering is purchased in order to know the customer and the purchase reference. The provided URL should support a POST request with the following structure.

```
POST notification_url HTTP 1.1
Content-Type: application/json
{
    "offering": {
        "organization": "CoNWeT",
        "name": "SmartCityLights",
        "version": "1.0"
    }
    "reference": "51c2d2825d9af944d0d1cfe0",
    "customer": "santander_crm"
}
```

9.3.5 **Accounting and Pay-Per-Use integration**

If a service provider wants to provide a service under a pay-per-use pricing model, it is necessary to develop some modules in charge of providing accounting info to WStore in order to allow it to perform the charging process. To perform the accounting process, the service provider must have received the purchase notification as defined in the previous section since the offering, the customer, and the reference are required.

The accounting request is defined as follows:

```
POST /api/contracting/{reference}/accounting HTTP 1.1
Content-Type: application/json
{
    "offering": {
        "name": "offering_name",
        "version": "1.0",
        "organization": "organization"
    },
    "component_label": "issues",
    "customer": "test_user",
    "correlation_number": "1",
    "time_stamp": "2013-07-01T10:00:00-0",
    "record_type": "event",
    "value": "1",
    "unit": "issue"
}
```
10 Business Modeler - User and Programmer Guide

10.1 Introduction

This document describes how to use graphical user interface of the business model editor and how to access business element resources using the REST API.

10.1.1.1 Background and Detail

This User and Programmers Guide relates to the Business Modeler GE which is part of the Applications and Services chapter. Please find more information about this Generic Enabler in the following Open Specification.

10.2 User Guide

The Business Modeler is a single page web app, using HTML5, CSS3 and JavaScript. It is a part of the Applications/Services ecosystem and provides a graphical tool to business experts for creating and evaluating high-level business models. The Business Modeler allows them to get an estimation of the profitability of a certain model and define value and money streams quickly and transparently.

The Business Modeler automatically starts with a new canvas to assemble your business elements and create a new business model.

Let's take a look at the user interface elements you're presented with:

![Business Modeler web app](image)

At the top you find the black menu bar and at the right a "Toolbox" sidebar. Business models and their elements are constructed using the Toolbox, let's focus on that first and come back to the menu bar later on.

10.2.1 Business model meta data

A business model can contain some relevant meta data, e.g. a title, a description and a list of authors. The Toolbox allows you to edit this information directly:
Click on the “Title” text field to start editing the title. Once you start editing the title a "Save changes" button appears. As you can guess it allows you to save your changes to the title, once you've finished editing it.

The same applies to "description" and "authors".

The Toolbox allows you to quickly edit the meta data of the business model, however if you want to do a more extensive edit you can click on the "Edit meta data" button.

This opens a dialog window which allows you to edit the meta data. You are presented with larger text fields and the "description" field can be enlarged if you want to write a more substantial description for the business model.
Meta Data Editor

"Cancel" will discard any changes you made and "Save meta data" will store your changes into the business model.

10.2.2 Time Range

A business model expresses values on a time scale. You need to define a time range for the business model. To edit the time range you need to click on "Edit Timerange" in the Toolbox.

Toolbox - Business Model - Buttons

You'll be presented with a dialog window which shows you the default time range starting at the current year and the two subsequent ones.
Time Range Editor

To add and remove years click on the "Add" and "Remove" buttons respectively. To edit years (maybe you want to increment the year value with two instead of one for instance) click on the year you want to edit.

Values:

- 2014
- 2015
- 2016

Click the buttons to add and remove values to the timerange.

Time Range Editor - Editing

Clicking "cancel" at the bottom of your window will discard your changes and "Save time range" will save them.

10.2.3 Business Elements

10.2.3.1 Actors

Actors represent a person or company in your business model. Click the "Actor" button to create one.
"Actor" - clicked

An Actor element is automatically placed on the canvas with a default "Actor (1)" name. If you click the Actor element a context-sensitive panel is added to the Toolbox.

"Actor" - selected

TIP: if the panel isn't entirely visible you can toggle the visibility of the other toolbox panels by clicking the "caret".
To edit the actors name click on the text field containing "Actor (1)" in the Toolbox.

![Actor editing](image)

Again, "Save changes" will store your changes.

The "Delete" button in the Actor panel will - as you might guess - delete the Actor element.

10.2.3.2 **Activities**

Activities are representations of what an actor "does". Activities are always linked to a specific actor. An actor can perform multiple activities, but each activity can only be performed by one actor. Due to this inherent connection between actors and activities all activities need to be related to an actor immediately. Click on the "Activity" button.

![Activity clicked](image)

You are presented with a dialog window which gives you a number of options to relate the new activity to an actor. If only one actor exists on the canvas it is automatically selected to allow you to relate the activity quickly to it. However when multiple actors exist you need to select the one you want to relate the activity to.
Once you select an actor you can click the "Assign Activity" button to make the relation between the new activity and the existing actor.

Or, in case you want to assign the activity to a new Actor you can click the "Assign to new Actor" button. This will create a new Actor and will automatically relate it with the new Activity.

Clicking "Cancel" will cancel the creation (and relating) of the a new Activity.

Just as with Actors if you click the new Activity element a context-sensitive panel will be shown in the Toolbox which allows you to edit its name and delete the element.

The Activity panel however has two buttons the Actor panel hasn't: "Switch Actor" and "Link to cost model". We'll come back the latter button later on. With "Switch Actor" you are again presented
with the above dialog window which allows you to assign the activity to one of the other actors. (The "Switch Actor" button is only shown when multiple actors are present on the canvas)

10.2.3.3 Flows

Revenue and product streams exist between activities, they are represented by "Flows" in the business modeler. By default the Toolbox has the "Monetary Flow" button selected.

You can create a flow by clicking on one of the small grey circles at the edges of an activity and dragging towards one of the grey circles at the edges of a different activity.

"Monetary Flow" - created

By clicking the flow again a context-sensitive panel is added to the Toolbox which allows you to edit the name of the flow, delete the flow and provides you with an overview what its origin and destination are.

"Monetary Flow" - selected
To draw Product flows click the "Product Flow" button in the toolbox first and then click-drag from and to the grey circles as with Monetary flows.

10.2.3.4 **Value Shares**

Value shares allow you to split a revenue stream between activities. You can create one by clicking the "Value Share" button in the Toolbox.

As with the other business elements a context-sensitive panel is shown in the Toolbox when you click on the newly created value share element.

This allows you to edit the value share name and to delete it.
You can create monetary flows (this will not work with product flows!) from an activity to a value share and vice versa just as you did between activities.

"Value Share" - flows created

10.2.4 Submodels

10.2.4.1 Cost, revenue and share models

Each of the business elements (except actors) can be linked to a submodel: activities to cost models, monetary flows to revenue models and value shares to share models. When you select a business element (e.g. an activity) and click on the "Link to cost model" button in the Toolbox you are presented with a dialog window which allows you to link the business element to a specific model.

Submodel - dialog window
Click on the dropbox with "-- Please select --" to see a list of the available submodels. Select one and click on "Assign cost model" to assign it to the activity. We'll come back to the "Required inputs", "Outputs" and "Assign and link inputs" later on.

Submodel - Cost model selected

The colour of the business element the submodel was assigned to changes from red to green to denote a submodel was assigned.

Business Element - colour change

In the Activity Toolbox panel this is represented too and on top of that it shows you the name of the assigned cost model.
10.2.4.2 **Input values and Model Values**

Submodels need data to be able to calculate the costs, revenues and shares. This data can be passed to the submodels through "input values". Each submodel has a predefined number of input values that need to be fulfilled to operate correctly. Model Values are values you declare for the business model which can be linked to these input values to feed the cost models with data.

We'll start with creating a model value. In the Toolbox, in the Business Model panel click on the "Edit model values" button.

Since there are no model values yet none are shown either. Click on "Create new Model Value". You will be asked for a "Name" and "Description".
Model Values Editor - Name and description

Provide a clear, concise and semantically meaningful name and click on "Next". Now you need to generate values using a time function.

Model Values Editor - Time functions

A value is needed for each year in the time range. We have provided you with a number of function to generate these values. Select a time function in the dropbox with "-- Please Select --".
Model Values Editor - Time function selected

Once a time function is selected you get an overview of how the time function will generate the values in the grey box next to the dropbox. Depending on the time function one or more values need to be provided to allow it to correctly generate a value for each year in the time range. Once you’ve provided all necessary values an overview of the to-be-generated values will be shown.

Model Values Editor - Time function values generated

Click on "Save" to create the Model Value. "Cancel" if you wish to do it later or "Previous" if you want to make any changes to the name and description. Now that a model value is created we can use them to provide the submodels with input data.

If you click on any of the business elements that is linked to a submodel you’ll see that the context-sensitive Toolbox panel has a “Link cost model inputs" button. When you click on it, you’ll be presented with the "Link input" dialog window.
On the left in the dropdown under "Model Values" you'll see the model value you just created and on the right you'll see the required input values for the assigned submodel. Once you've selected a model value you can click on the small button with a chain in front of an input value to link the model value to the input value.

Once a model value is linked to an input value the corresponding representation on the canvas will turn green and the context-sensitive Toolbox panel will show it too.

The menu allows you to perform some standard operations on the business model.

With the File menu you can create a new business model or save the current one to your hard-drive. Subsequently you can open a business model file.
When you click on "Save" you'll be presented with a OS-specific file system browser.

After selecting the location you want to save the XML file to, click on "Save".

To open a business model file click on "Open" in the file menu. You'll be presented with a dialog window which allows you to select the business model file you want to open.
Open a business model

First click on "Choose files" and navigate to the location of the business model file you want to open using the OS-specific file system browser.

File system browser

Then click on "Open". You'll see the name of the file you chose in the "File input" field.

File input

Choose Files SingleActor.xml

Please select a valid business model document.

Open business model - file selected
Now click on "Open document" to open the business model file in the editor.

10.2.5.2 **Edit**

With the "Edit" menu you can undo/redo your actions.

For instance, if your last action was the addition of an activity clicking "undo" will remove the activity. If you change your mind again, you can click on "redo" to re-add it to the canvas.

10.2.5.3 **View**

The "View" menu allows you to control how the canvas is shown.

With "zoom in" you can zoom in on the canvas, showing the elements bigger.
"zoom out" does the opposite and "1:1" will restore everything to their usual size.

10.2.5.4 **Snap**

The "Snap" menu presents you with options to align your business elements on the canvas.

"Grid" will render a grid on the canvas. This allows you to snap business elements on fixed locations when moving them. Just click-drag an element to have it snap to the grid.
"Object" allows you to align elements to other elements and "No snap" turns off all snapping features.

10.3 Programmer Guide

The business modeler provides programmers with a REST interface for retrieving submodels.

10.3.1 CMD

The URL endpoint for the CMD (calculator model document) is /api/cmd/models. For example, if you installed the Business Modeler to run on the default host and port you need to use the following url: http://localhost:9100/api/cmd/models

10.3.1.1 In general

All API calls accept some common parameters.

10.3.1.1.1 Accept Header

The Accept-header allows you to define in which format you want the data to be rendered.

- Accept: application/xml
- Accept: application/vnd.iminds.bme+xml
- Accept: application/json
- Accept: application/vnd.iminds.bme+json

Will return all data in XML-format.

Will return all data in JSON-format.

10.3.1.1.2 Format extension

Adding .json or .xml to any API call will again define the format of the returned data.

For example:

- /api/cmd/models.json
10.3.1.1.3 Download parameter

If you add download or in short-hand dl as a query string parameter, the result of the API call will automatically be presented as a download. (I.e. with a Content-Disposition: attachment header)

For example:

- /api/cmd/models?dl

10.3.1.2 GET /api/cmd/models

Will provide you with a list of all submodels. By default this is returned as JSON.

```json
{
  "CMD": {
    "model": [
      {
        "id": "firstBPMN",
        "subType": "bpmn",
        "type": "cost"
      },
      {
        "id": "secondBPMN",
        "subType": "bpmn",
        "type": "cost"
      },
      {
        "id": "oneAdditionalModel",
        "subType": "ecmn",
        "type": "cost"
      },
      {
        "id": "test",
        "subType": "ecmn",
        "type": "cost"
      },
      {
        "id": "first revenue model",
        "subType": "revenue",
        "type": "revenue"
      }
    ]
  }
}```
Or, when specifically asked using one of the above formatting possibilities, in XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cmd:CMD xmlns:cmd="http://specs.iminds.be/cmd/0.5">
  <cmd:model id="firstBPMN" type="cost"/>
  <cmd:model id="secondBPMN" type="cost"/>
  <cmd:model id="oneAdditionalModel" type="cost"/>
  <cmd:model id="test" type="cost"/>
  <cmd:model id="first revenue model" type="revenue"/>
</cmd:CMD>
```

10.3.1.3 **GET /api/cmd/models/<id>?subtype=<subtype>**

By passing an id as part of the path to the models API call and the subtype as a query string parameter you can retrieve a specific model.

For example:

- GET /api/cmd/models/firstBPMN.xml?subtype=bpmn

Will return:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cmd:CMD xmlns:cmd="http://specs.iminds.be/cmd/0.5">
  <cmd:model id="firstBPMN" type="cost" name="firstBPMN"/>
</cmd:CMD>
```

- GET /api/cmd/models/first%20revenue%20model.xml?subtype=revenue

returns:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cmd:CMD xmlns:cmd="http://specs.iminds.be/cmd/0.5">
  <cmd:model id="first revenue model" type="revenue">
    <cmd:description>The first revenue model contains some example model for the revenue</cmd:description>
  </cmd:model>
</cmd:CMD>
```

- GET /api/cmd/models/test.xml?subtype=ecmn

Returns:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
<cmd:CMD xmlns:cmd="http://specs.iminds.be/cmd/0.5">
   <cmd:model id="test" type="cost">
      <cmd:inputs/>
   </cmd:model>
</cmd:CMD>
11 Business Calculator - User and Programmer Guide

11.1 Introduction

This document describes two different aspects of making use of the business calculator GE.

1. How to make use of the business calculator as a graphical user interface which is capable of simulating your costs and revenues for a given scenario. The business calculator provides a web-based graphical user interface aimed at drawing out the cost structures into the different domain specific languages as described in the open specifications (see link in background) and links these to the calculator/simulator backend making it into a complete stand-alone toolkit.

2. The necessary steps to develop a software application or another user interface which makes use of the business calculator back-end functionality. This interface is actually already incorporated into the interaction the business modeler GE has with the business calculator. As such the business calculator API provides for a backend interaction mode and is based on REST principles and generally accepts and returns XML encoded messages. Since both REST as XML are independent from a concrete programming language, you just have to know how to make an HTTP request and write/parse markup textual data in the programming language of your choice.

11.1.1 Background and Detail

This User and Programmers Guide relates to the Business Calculator GE which is part of the Architecture of Applications and Services Ecosystem and Delivery Framework Applications and Services chapter. Please find more information about this Generic Enabler in the following FIWARE.OpenSpecification.Apps.BusinessCalculator.

11.2 User Guide

11.2.1 Goal of the Business Calculator and its different parts

The Business Calculator uses the input from the Business Modeler to dig deeper into the cost structure of certain business activities. By allowing the user to insert extra parameters and a visualized representation of the underlying model in the editor, more detailed analysis of the time-evolution of the activity's cost structure can be calculated by the calculator.

Depending on the type of activity, a different business calculator can be invoked. Currently, two types are implemented: the Equipment Cost Modeling Notation (ECMN) and Business Process Modeling Notation (BPMN). Both consist of a graphical user interface (the editor) and a backend simulator (the calculator).

This document will shortly describe how to use both editors, by first detailing general characteristics (common for both types), to then further describe the specifics of each editor.

11.2.2 General characteristics of the editors

The editor is built in a modular fashion in which different functionalities are linked to each other to make the interaction tuneable for the user, allowing different domain specific languages and stay as intuitive as possible. In this section we describe the way the generic building blocks are implemented and we detail how the user can access their functionality. In the following sections, we will describe the additions and deviations in the different domain specific parts of this interface. We follow the same structure as the general section, running through the different windows, but only indicate specifics when deviating from the general description. When a section is not included, this means that no additions are made on top of the generic implementation as indicated above.
The editor consists of following parts:

- the **menu box** (1), which includes typical loading, saving, calculating and layout options,
- the **palette and command line** (2), to draw the model graphically or through code specifications,
- the **canvas** (3), which visualizes the model,
- the **properties field** (4), indicating the characteristics of the selected element,
- the **output field** (5), showing graphical representations of the calculated results
- the **comments field** (6), which allows the user to add comments, remarks or suggestions on the model or one specific element thereof

### 11.2.2.1 The menu box

The menu box is situated at the top of the editor, and includes the main features of every standard application. We give a short overview of the necessary features:

- **Model tab**
  - **Load model**: load a previously saved model
  - **Create new model**: start developing a new equipment model
  - **Save model**: save the current model
  - **Export to pdf**: allows the user to save the visualization, properties and/or selected output as pdf in a chosen location

- **Calculator tab**
  - **Calc**: Calculates the entire model (making the link with the calculator), and shows the default results (see further) on the right hand side

- **View tab**
  - **Zoom**: opens a zooming box, which allows the user to set the zooming percentage manually
Future Internet Core Platform

- **Window**: gives an overview of all windows available in the editor and allows you to select/deselect
  - *Perspective*: there are several perspectives to switch a predefined window-positioning into place

- **TimeFunction**: The time-function tab allows to define a new trace of input values to be linked to parts of the calculations
  - **new timefunction**: make a new time-function
  - **edit existing timefunction

- **Editor tab**
  - **Switch editor**: allows to switch between the different types of editors and the different domain specific languages linked to that

### 11.2.2.2 The palette and command line

The editor should be straight forward to use, both for technically schooled as well as non-technically schooled people. We therefore foresee a graphical drawing palette, as well as a debugging command line field on which you currently see the commands issued and can check what is happening. This command line is also a placeholder for a CLI in which the model can be changed by means of commands instead of by means of point and click.

The palette gives an overview of all objects that can be added to the model. The user selects one object, goes to the right position on the screen (canvas see further) and clicks there to place one element of that object type. If the user wants to place a second element of the same object type, he should just go to the right position on the screen (there should be no need to re-select the same element again). If the drawing palette is used for adding/editing elements, the command line still shows the effect in code.

### 11.2.2.3 The canvas: the visualization of the model

In the middle of the screen, the actual visualization of the model is shown in the model canvas. It contains the different elements and their linkages. When you select element, which then becomes highlighted, the properties field tab shows the specifications of the element, while the output field tab shows the default output for that specific element (e.g. for an equipment type, the output field would show its cost over the years in a line graph).

The model can consist of submodels, which are indicates by a small “+” in the right bottom corner. The submodel is shown in a pop-up window when clicking on the “+”, returning to the original model is done by closing the submodel window (or minimizing it). Linking two elements in the canvas can be done in two distinct ways:

- **“Select connector – click - click”**
  Move with your arrow to the connector in the palette, select the connector (which is highlighted), move to the first element in the canvas, click to select the first element (is highlighted), move to the second, click. The connector now appears, together with its property window.

- **“Ctrl + click - click”**
  While pressing the CTRL button, move with your arrow to the first element in the canvas, click to select the first element (which is highlighted), move to the second, click. The connector now appears, together with its property window.

### 11.2.2.4 The properties window

When an element in the model canvas is selected, the properties field appears on the right hand side of the screen. This field allows to check and edit the different parameters of the selected element, which depend on the type of the element.
11.2.2.5 *The output field*

The output field shows all results of the calculations in three possible formats: a line graph, a bar graph or a textual column based output. The output is easily inspectable by means of a selection subpanel allowing the user to select the traces to show or select traces which are linked to a given input-trace (driver, cause,...). The values of a result are shown when hovering over the result graph.

11.2.2.6 *The comments field*

The last field of the editors is the comments field, in which the user can indicate remarks, comments or suggestions. A comment can address the entire model (or submodel), or can focus on one specific element.

A comment thus consists of three fields:

- Element or model, which can be filled in the comments field manually, or can be filled automatically by selecting the right element,
- User, who enters the comment,
- The comment itself.

11.2.3 The ECMN editor

Apart from the general look-and-feel and characteristics described in the previous section, the specifics of the ECMN editor will be detailed in this section.

11.2.3.1 *The palette and command line*

The basic building blocks of the equipment model are drivers, equipment, submodels, connectors, aggregators and separators. For more information about the specificities of each building block, we refer to the Open Specifications document.
11.2.3.2 The properties window

The properties window shows the specific properties of the highlighted element in the model canvas. Depending on the type of element, the properties differ:

- **Equipment**
  - **Name**: identifier, can be changed, not necessarily unique
  - **ID**: unique identifier of the equipment, cannot be changed
  - **Life time**: if the equipment needs to be replaced after a certain amount of time, it can be specified here
  - **Valid period**: in case of migration to a newer system, checking this box allows to use this type of equipment only for a limited period in the calculation horizon

- **Driver**
  - **Name**: identifier, can be changed, not necessarily unique
  - **ID**: unique identifier of the driver, cannot be changed

11.2.3.3 The output field

The output field allows to switch between amounts installed and costs of the installation base by means of a button appearing on the top left of the window.

11.2.4 The BPMN editor

11.2.4.1 the Menu box

One additional menu item has been added to the menu box: resources, which allows to define resources consumed in a task in the operational process. This opens up a wizard which allows to add a new resource and define the characteristics of this resource.

11.2.4.2 The Palette

The BPMN editor has a simplified palette for the full BPMN modeling language and works in the same manner as the ECMN modeling approach.

The basic building blocks of the business process model are events, tasks, submodels, connectors, gateways. For more information about the specificities of each building block, we refer to the Open Specifications document.

11.2.5 The Revenue editor

11.2.5.1 the menu box

In the menu box the link to a schematic graph of the model has been added.
11.2.5.2 **The Canvas and Palette**

The canvas and palette have been replaced by a wizard style description of the model. The wizard allows to detail one part of the revenue model by a rule and there are buttons on the right of the rule to (in order):

1. add a sibling rule that will be checked if this rule is not applicable (this is not available on the root as there is only one root)
2. add a child rule, which will be checked if this rule is applicable. Typically multiple child rules are defined and a choice is made between them based on their condition.
3. remove this rule and all its children

Additionally the wizard allows to define how the revenue should be calculated when the constraints are met. Finally the wizard also allows to calculate the revenues for a set of test-traces.

11.2.6 The hierarchical network architecture

11.2.6.1 **The canvas and palette**

In this domain specific language again the canvas and palette have been replaced by a wizard style description of the model. The wizard allows to detail how the area or connection points are linked to connection locations (for instance, to a central office or a router location). This is again in the same manner possible in a hierarchical manner allowing to split the area in subareas which again connect to subareas and so on, finally connecting the most fine grained connection points (e.g. customers, sensor nodes, ...). These finer grained approaches are linked by means of a child node. Additionally the hierarchical network structure also allows to define alternative network architectures next to each other by means of sibling nodes. Finally at each level the user can select how the connection should be made to the lower level. More details can be found in the open specifications.
Finally the wizard also allows to calculate the revenues for a set of test-traces.

11.3 Programmer Guide

11.3.1 The Business Calculator

The business calculator packs a set of cooperating services which should be installed together. when all are installed on the same server the editor will automatically link correctly the different elements of the installation. In order for the business calculator to allow an easy linking to the business modeler and possibly to later installations of additional calculators as well, the business calculator hub has been added to the process. This hub does exactly what its name suggest, act as a one point of contacting the right calculator for your need, but installation of such hub has other advantages as well:

- One uniform REST interface for calling the calculator (more on this later)
- Plugin and out of calculators which allows for
  - easy installation of new calculators
  - load balancing with multiple calculators
  - run-time linking of the calculator in case of failures
- Maintain compatibility of versions of calculators and storage formats over time
- Inherently a modular design with a split of responsibility and isolating the functionalities of the calculators

The full programmer interface and the usage is shown in the figure underneath:

The REST Interface will be depending on the server name and location, but can be linked at installation time into the editor. The configuration files of the editor will automatically select the standard installation on the same server as standard location for the services. The calculation is handed over to the calculator hub which will detect the type of request and the type (and version for future use) of model and relay this call to the right calculator. Again in the standard installation the location of the servers is implicit and no additional settings should be managed for this. The REST interface called is:
PUT name of the server/CalculatorHub/rest/service/calc
with in the message body the data to be used for the calculations (more information on this below)

All interaction between an external application and the Business Calculator will run over this REST interface and all information conveyed over this information link makes use of an integrated XML format. The XML format contains the model to be used for the calculation step and the additional context to be linked to the model for its calculation. This data consists of the following three parts:

1. model
2. inputs
3. outputs

More details on the main XML document and the subsequent formats are given in the subsections

11.3.1.1 The Calculation XML format

All data for a calculation is encapsulated inside a calculation tag and contains in a structured manner the data for the model, the inputs and potentially the outputs as well. As such a calculation is a fixed format and stores every step in the process at once, and can be triggered and adapted from everywhere without a need to delve in the details of the format.

The calculation contains details on the following three parts:

1. the model to calculate
2. the inputs to calculate the model with
3. the outputs of the calculation

The XML format is the following:

```
<calculation version="1.0">  
  <model id="name" type="type of model" version="1.0">  
    more details underneath  
  </model>  
  <inputs>  
    <input name="name in the model" id="link to model">  
      <start>value</start>  
      <years>value</years>  
      <values>space separated values</values>  
    </input>  
  </inputs>  
  <outputs>  
    <output name="name" id="link to model">  
      <start>value</start>  
      <years>value</years>  
      <values>space separated values</values>  
    </output>  
  </outputs>  
</calculation>
```
The XML format for the model will be discussed in more detail underneath for the different types of models. There is an identification of the id of the model (which can be linked to a unique model-id on the repository) as well as the type and version of the model format. This type and version will be used to select the right editor and right calculator for the model.

The inputs and outputs are all defined in terms of time dependent functions. The inputs are time-dependent functions and are defined in terms of their starting year, the number of years they run and their values. In this way all calculations can be automatically trimmed to the right scope of years over which data is available. Both the inputs and the outputs are linked to the elements in a model and as such an ID to this element (if existing) in the model is provided.

11.3.1.2 The ECMN XML format

An ECMN model is stored in an XML format in the repository in the same format as it is send to the calculator. In the calculation tag, the type is indicated as "ECMN".

The ECMN model contains all data on the ECMN model and the structure is shown below:

```xml
<model type="ECMN" id="modelname">
  <equipments>
    <equipment id="modelname_equipment_number" ypos="" xpos="" name="name">
      <capacity>value</capacity>
      <cost>value</cost>
      <size unitType="empty/name">empty/value</size>
      <maintenance>value</maintenance>
      <energyConsumption>value</energyConsumption>
      <installation>value</installation>
      <floorspace>value</floorspace>
      <recurrentInstallation>value</recurrentInstallation>
    </equipment>
  </equipments>
  <drivers>
    <driver id="modelname_driver_number" ypos="" xpos="" name="name">
      <recurrentInstallation>value</recurrentInstallation>
    </driver>
  </drivers>
  <connectors>
    <connector id="modelname_connector_number" ypos="" xpos="" name="name example normal granularity">
      <srcId>id</srcId>
      <dstId>id</dstId>
      <granularity granularityType="1">
        <srcGranularity>value</srcGranularity>
        <dstGranularity>value</dstGranularity>
      </granularity>
    </connector>
  </connectors>
</model>
```
<inDrivers>
  <inDriver>
    <subModelInDriverId id="id to use in the overarching model to link to" ypos="" xpos="" name="name"/>
    <linkDriverId>id of the original model driver to link to</linkDriverId>
  </inDriver>
</inDrivers>

11.3.1.3 The BPMN XML format
The BPMN model format is simply the XPDL model format encapsulated inside a model tag

<model type="BPMN" id="modelname">
  ... here XPDL content
</model>

11.3.1.4 The Revenue Model XML Format
The revenue model is encapsulated in a model tag with as type revenueModel or revenue and with further content as defined below:

<model version="1.0" type="RevenueModel" id="name">
  <parameters>
    <param type="not-set" id="revenue-model" fixed-template="false">
      <childParams num-of-child-id="2">
        <param type="time-base" id="revenue-model.timeperiod1" fixed-template="false">
          <attributes>
            <attribute value="1" name="year-start"/>
            <attribute value="5" name="year-end"/>
          </attributes>
          <pricingScheme type="limit-pricing">
            <pricing-attrs>
              <attribute value="5" name="time-span"/>
            </pricing-attrs>
            <profit fixed="false"/>
11.3.1.5 The Hierarchical Network Structure Model XML Format

The hierarchical network structure model is stored in a model tag with the type "PNMN" referring to the physical network modeling notation, which was initially used but is no longer in use as the official name. Two flavors of the PNMN format exist, which correspond to the two ways of building them in the editor: top-down and bottom-up and a distinction between both is already made in the model tag as well. An example of both can be found below:

11.3.1.5.1 top-down format

```xml
<Model id="name" type="PNMN" pnmnType="Top-down" version="1.0">
```

<node id="root">
  <ExtendedAttributes>
    <ExtendedAttribute Name="nameOfElement" Value="name"/>
    <ExtendedAttribute Name="numOfElementShouldBeInstalled" Value="value>1"/>
    <ExtendedAttribute Name="numOfSurfaceToBeCovered" Value="value>0"/>
    <ExtendedAttribute Name="surfaceType" Value="km2 area covered|# connection points"/>
    <ExtendedAttribute Name="analyticalModel" Value="Street Length|Simplified Street Length|Double Street Length|Diagonal Tree|Simplified Steiner Tree"/>
  </ExtendedAttributes>
  <childNodes>
    <node id="root.1">
      <ExtendedAttributes>
        <ExtendedAttribute Name="nameOfElement" Value="name"/>
        <ExtendedAttribute Name="numOfElementShouldBeInstalled" Value="value>0"/>
        <ExtendedAttribute Name="numOfSurfaceToBeCovered" Value="value"/>
        <ExtendedAttribute Name="surfaceType" Value="km2 area covered|# connection points"/>
        <ExtendedAttribute Name="analyticalModel" Value="Street Length|Simplified Street Length|Double Street Length|Diagonal Tree|Simplified Steiner Tree"/>
      </ExtendedAttributes>
      <childNodes/>
    </node>
  </childNodes>
</node>

11.3.1.5.2  bottom-up format

<Model id="test" type="PNMN" pnmnType="Bottom-up" version="1.0">
  <node id="root.1">
    <ExtendedAttributes>
      ... same as in top-down
    </ExtendedAttributes>
  </node>
</Model>
11.3.2 A Limited Repository

For storing and exchanging models between applications, the business calculator contains a very small repository implementation. This repository is used for the sake of prototyping and integrates with the front-end in a straightforward way. It exposes the following REST interfaces to programmers to get the list of available models and the details of a model of the server. The interfaces are not streamlined to each-other, but this will clearly be an aspect in the valorization steps of the tool.

11.3.2.1 ECMN models

GET server name/ECMNCORE/rest/model/username/getallmodelid

GET server name/ECMNCORE/rest/model/get/modelname

11.3.2.2 Other

GET server name/Repository/rest/service/username/getallmodelid/modeltype=bpmn|revenue|pnmn

GET server name/Repository/rest/service/get/modelid=modelname&modeltype=bpmn|revenue|pnmn