

**Private Public Partnership Project (PPP)**

**Large-scale Integrated Project (IP)**



fi-ware

**D.7.4.2: 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-WP7-D.7.4.2

**Project Document Date:** 2013-04-30

**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 "Interfaces to Network and Devices" (I2ND) chapter. The document also details the required software elements and the procedure to setup an environment for the development of applications that can exploit the capabilities of the I2ND Generic Enablers.

This document consolidates new contents and also contents in previous issues of Release 1. The reason for re-delivering parts that were already issued is twofold:

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

## 1.2 About This Document

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

## 1.3 Intended Audience

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

## 1.4 Chapter Context

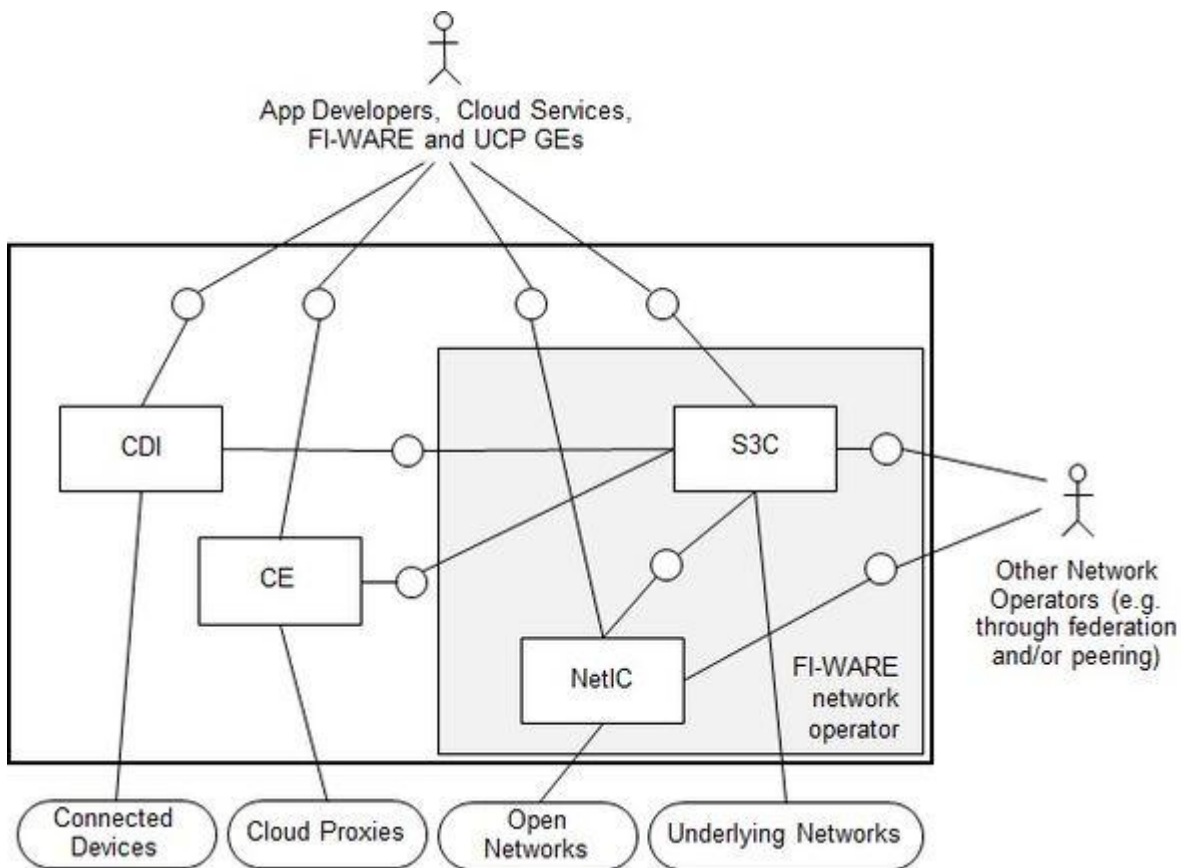
The I2ND chapter defines an enabler space for providing Generic Enablers (GEs) to run an open and standardised network infrastructure. The infrastructure deals with highly sophisticated terminals, as well as with highly sophisticated proxies, on one side, and with the network operator infrastructure on the other side. This latter will be implemented by physical network nodes, which typically will be under direct control of an operator, or the node functionality will be virtualised – in this case the I2ND functionality can be accessed by further potential providers, like virtual operators. The I2ND chapter defines four GEs, as represented in the figure:

- CDI (Connected Device Interface) towards the Connected Devices. These devices include, but are not limited to, mobile terminals, tablets, set top boxes and media phones.
- CE (Cloud Edge) towards the Cloud Proxies. Cloud Proxies are gateways, which will connect and control a set-up of nodes towards the Internet or/and an operator network.
- NetIC (Network Information and Control) towards Open Networks. Open Networks are following the idea of flow based controlled networks, and can be used for virtualisation of networks.
- S3C (Service Capability, Connectivity and Control) towards Underlying Networks. The underlying networks are following standards such as Next Generation Networks (NGNs) or Next Generation Mobile Networks (NGMNs). In the case of the S3C specified in I2ND the baseline underlying network will be the Evolved Packet Core (EPC) by 3GPP.

Each of the GEs of I2ND have specific interfaces which can be accessed by Application Developers, Cloud Services, FI-WARE and third party Enablers. Besides typical interfaces to functionality offered by such GEs, it is worth mentioning that:

- The GE S3C is the central point of the I2ND architecture. I2ND develops an enabling environment which can be used by network operators. Together with NetIC, both GEs build the environment of an operator, which might even be a virtual operator. S3C can be seen as the GE to run and steer the network infrastructure.
- The interfacing between S3C and CDI provides status and control information exchange of the device and remote control capabilities.

- Cloud proxies can be part of an operator infrastructure. Therefore it is necessary to have access to these network nodes through a standardised interface.



## 1.5 Structure of this Document

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

The following resources were used to generate this document:

### D.7.4.2 User and Programmers Guide front page

[Connected Device Interfacing - User and Programmers Guide](#)

[Cloud Edge - User and Programmers Guide](#)

[Network Information and Control - User and Programmers Guide](#)

[NetIC GE - OFNIC Uniroma - User and Programmers Guide](#)

[Service Capability Connectivity and Control - User and Programmers Guide](#)

[S3C GE - OpenEPC - User and Programmers Guide](#)

[S3C GE - SMS+MMSEnablers - User and Programmers Guide](#)

[S3C GE Support - S3C API Proxy - User and Programmers Guide](#)

## 1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

### 1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.

Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

### 1.6.2 Figures

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

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

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

### 1.6.3 Sample software code

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

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

## 1.7 Acknowledgements

The following partners contributed to this deliverable: [ALU](#), [DT](#), [FRAUNHOFER](#), [FT](#), [INTEL](#), [NSN](#), [TRDF](#), [TI](#), [UNIROMA1](#)

## 1.8 Keyword list

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

## 1.9 Changes History

Release	Major changes description	Date	Editor
v1	First draft of deliverable submission generated	2013-04-22	TID
v2	Initial version with GE pages added	2013-04-22	TI
v3	Updates to links, ready for finalisation	2013-05-20	TI

## 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.....	6
1.8	Keyword list .....	6
1.9	Changes History .....	6
1.10	Table of Contents .....	6
2	Connected Device Interfacing - User and Programmers Guide .....	9
2.1	Introduction.....	9
2.2	Users Guide.....	9
2.2.1	CDI's implementation architecture .....	9

- 2.2.2 Software and Hardware environment..... 10
- 2.3 Programmers Guide..... 10
  - 2.3.1 Including Libraries..... 10
  - 2.3.2 Library Initialization ..... 11
  - 2.3.3 CDI API Discovery ..... 11
  - 2.3.4 CDI API example usage..... 12
- 3 Cloud Edge - User and Programmers Guide..... 19
  - 3.1 Introduction..... 19
    - 3.1.1 Goal of this document ..... 19
  - 3.2 Programmers Guide..... 19
    - 3.2.1 Create a CloudEdge Virtual Machine image for STB (HTTP web server)..... 19
    - 3.2.2 Run a CloudEdge Virtual Machine image on STB (HTTP web server) ..... 21
    - 3.2.3 Run others CloudEdge Virtual Machines on STB..... 21
- 4 Network Information and Control - User and Programmers Guide ..... 22
- 5 NetIC GE - OFNIC Uniroma - User and Programmers Guide..... 23
  - 5.1 Introduction..... 23
  - 5.2 User Guide ..... 23
  - 5.3 Programmer Guide ..... 23
    - 5.3.1 Navigation with browser..... 23
    - 5.3.2 OFNIC Uniroma GUI (*draft version*)..... 26
    - 5.3.3 Apache HttpClient..... 27
    - 5.3.4 How to extend OFNIC Uniroma GE ..... 29
- 6 Service Capability Connectivity and Control - User and Programmers Guide..... 30
- 7 S3C GE - OpenEPC - User and Programmers Guide ..... 31
  - 7.1 Introduction..... 31
  - 7.2 User Guide ..... 32
  - 7.3 Programmer Guide ..... 33
- 8 S3C GE - SMS+MMSEnablers - User and Programmers Guide ..... 34
  - 8.1 Introduction..... 34
    - 8.1.1 Restrictions..... 34
    - 8.1.2 Countries, Short Codes and sender addresses..... 34
  - 8.2 User guide ..... 35
    - 8.2.1 1/Pre-requisites..... 35
    - 8.2.2 2/Sending your first SMS message ..... 35
  - 8.3 Programmer guide ..... 36
- 9 S3C GE Support - S3C API Proxy - User and Programmers Guide ..... 37

9.1	Introduction.....	37
9.2	User guide .....	37
9.3	Programmer guide .....	37



## 2 Connected Device Interfacing - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

### 2.1 Introduction

This document provides a useful guide for developers and users of web applications written on top of the CDI API. Specific sections are dedicated to developer tools intended to help them during development, test and deployment of CDI based Webapps. These include illustrations demonstrating how to discover and use the CDI APIs.

CDI installation and administration information can be found at the [Installation and Administration Guide](#) page.

### 2.2 Users Guide

CDI (Connected Device Interface) is a set of common JavaScript APIs which exist on a range of platforms, allowing developers to create web applications. CDI's common API set provides application portability. CDI is implemented as a JavaScript library with additional native software components. CDI's JavaScript library `cdi.js` can be used either by its self inside a normal web page, or inside a more complete CDI deployment which includes that native software components.

CDI provides a wide range of functionality via a big API. The API is broken into functional blocks. A Functional block is a group of APIs which provide a given set of functionality, e.g. Geolocation would be one functional block, and File System Access would be another. CDI can be deployed on a wide range of devices. Each device offers different functionality, not all functional blocks may be available on all devices, therefore CDI provides a method of detecting the presence of a functional block. A complete list of all of the available functional blocks can be found [in the FI-WARE Architecture Description](#). When the `cdi.js` is used without the native software components it provides less functional blocks than would otherwise be available.

#### 2.2.1 CDI's implementation architecture



CDI is implemented on top of [Webinos](#), a platform developed within another FP7 Project. The native software components are implemented as Webinos extensions and provide functionality which requires access to native, or host operating system interfaces. CDI provides a single JavaScript object `$cdi` through which all functional blocks and their APIs are available.

### 2.2.2 Software and Hardware environment

For the FI-WARE 2nd major release CDI supports Android devices, with both the `cdi.js` JavaScript library and native software components. The JavaScript library works independently of the native software components. It can be used as a JavaScript library within a web site / web page and provides support on iOS, Android, and PC platforms.

## 2.3 Programmers Guide

CDI makes it possible for a programmer to create a web page with rich functionality and deploy it as a hosted Webinos application with enhanced functionality. To do this all CDI based applications must be structured in the same way. Firstly the programmer must include the Webinos and CDI libraries, then wait for the CDI Library to Initialize, lastly the programmer must validate that the functional block containing the functionality they wish to use is available. These three steps are covered in detail below.

A CDI based application can be developed and deployed in 3 ways:

1. *As an HTML document (with no native software)*: The application is a HTML5 web page which does not use any of the enhanced native features of CDI.
2. *As an HTML document (with native software)*: A web page which connects to and uses a Webinos core located on the viewing device. Here the application exploits the Webinos messaging system to convey JavaScript calls out of the browser, toward the Webinos PZP. This is achieved by using the HTML5 WebSockets API, that implies the Browser must support the W3C WebSockets specification. WebSockets is fully supported by Chrome and Firefox, however you can refer this [page](#) to check WebSockets compliancy within your browser.
3. *As a Webinos widget*: If the application is packaged as a Webinos widget, it will be executed in the WebRT provided by Webinos. Webinos widgets are installed by opening the Webinos-Application activity (that comes with the apks installation) and scanning the SD-Card of your device looking for `*.wgt` files. A specific section is dedicated to the packaging and installation of Webinos widgets.

### 2.3.1 Including Libraries

CDI based applications are web applications, and are therefore HTML documents. To use CDI with the native software components it is necessary to import the `webinos.js` file and the `cdi.js` files. It is important that they are imported *in that order* (`webinos.js` `<code>` THEN `<code>` `cdi.js`).

```
<html>
  <head>
    <!-- ... -->
```

```
<script type='text/javascript' src='webinos.js'></script>
<script type='text/javascript' src='cdi.js'></script>
<!-- ... -->
</head>
<body>
  <!-- body here -->
</body>
</html>
```

### 2.3.2 Library Initialization

The CDI framework performs a number of initialization steps when it loads and is not immediately available for use. Any code which wishes to make use of the CDI API must wait until it is available. This is possible by waiting for the `onCDIReady` event:

```
window.addEventListener("onCDIReady", function(evt) {
  // ... make use of CDI APIs here
}, false);
```

This event always happens after the document is ready and has been loaded. It therefore supersedes these existing DOM events.

### 2.3.3 CDI API Discovery

CDI can work across a number of different types of devices and each different device can offer a different set of functionality. To cope with this CDIs APIs are broken down into functional blocks. It is possible to query to see if a device supports a specific functional block. The code example below shows how to check for the presence of the QoE and Geolocation APIs:

```
window.addEventListener("onCDIReady", function(evt) {
  if ( $cdi.qoe.availability == true ) {
    // .. use QoE API here
  }

  if ( $cdi.sensors.availability.geolocation == true ) {
    // ... use the Geolocation API here
  }

}, false);

}, false);
```

More detailed info about the API are available in the OpenSpecs document.

### 2.3.4 CDI API example usage

This section is aimed at providing a complete example of a Webapp based on CDI. There are two examples, one focused on the QoE functionality, and the other demonstrating a simple Geolocation use case.

**Note** the following examples use the jQuery library. Developers are then supposed to import jQuery before of executing the code.

#### 2.3.4.1 *Basic Web Application : QoE*

The tutorial will show how the QoE API can be used to control the user's experience for a video stream application. This example shows how the QoE API works for a video streaming application that wishes to offer QoE support in its UI. The following code can be run on top the local web Browser available in your device, and relies on a running PZP instance equipped with the FI-WARE QoE API Handler module.

Please be aware that QoE functionality is only available when creating a CDI application which uses native software functionality. QoE only works when CDI runs with Webinos and the QoE core is an Android Java package which is only accessible via a Webinos extension.

The following html file will be named as `cdi-qoe-helloworld.html`.

```
<html lang="en">
<head>
  <meta charset='utf-8'>
  <title>HTML5 Video and QoE API</title>
  <script type="text/javascript" charset="utf-8"
src="webinos.js"></script>
  <script type="text/javascript" charset="utf-8"
src="cdi.js"></script>
  <script type="text/javascript">
    var media_events = new Array();
    media_events["play"] = 0;
    media_events["pause"] = 0;
    media_events["ended"]=0;

    var flowid;
    var flowurl;
    var ended=false; //end of the user experience

/*Called when the document is loaded*/
function init() {
```

```

    document._video = document.getElementById("video");
    init_events();
}
function init_events() {
    for (key in media_events) { //Register listeners for
video events
        document._video.addEventListener(key, capture, false);
    }
}
document.addEventListener("onCDIReady", init, false);
/*Called to update the UI with fresh context info*/
function onCtxtChanged(msg) {
    $cdi.qoe.getContext(flowid,
        function(msg) {
document.getElementById("status").innerHTML="Monitoring status:
"+msg.ctxtdump.monitor;

document.getElementById("feedback").innerHTML="Click Rate:
"+msg.ctxtdump.CR;

        },
        function(errmsg){});
    //refresh the context each 1 second
    if(!ended) {setTimeout(onCtxtChanged,1000);}
}
/*Handle video events*/
function capture(event) {
    if(event.type=="play"){
        flowurl= document.getElementById('mp4').getAttribute('src');//get
the video URL
        $cdi.qoe.bindFlow(flowurl, //bind the video URL to the QoE
framework
        function(msg) {
            flowid=msg.flowID;

document.getElementById("flowid").innerHTML="Assigned flow ID: "+flowid

        $cdi.qoe.runQoeMonitor(flowid,onCtxtChanged,function(errmsg){});},
            function(errmsg){alert("An error occurred while
binding the flow: "+'\n'+errmsg.bcerr.code_desc);});

```

```

    }
    else
    if(event.type=="pause"){

        $cdi.qoe.pauseQoeMonitor(flowid,onCtxtChanged,function(errmsg){});
    }
    else
    if(event.type=="ended"){
        ended=true;

        $cdi.qoe.unbindFlow(flowid,function(msg){alert("end of the
experience");},function(errmsg){});

/*Attached to the QoE feedback button*/
var feedback=function(){
    $cdi.qoe.giveFeedback(flowid,onCtxtChanged, function(errmsg){});
}
</script>
</head>
<body>
    <h1>QoE API and HTML5 video example</h1>
    <p>This page is a demonstrations of the QoE API applied to a video
streaming application. When the video is played,
        its URL is bound to the QoE monitoring system, and a numeric
FlowID is generated.

        A button to provide bad QoE feedbacks is provided, together with
fields to track a subset of the flow's context</p>
    <div>
        <video id='video'
controls preload='none'
poster="http://media.w3.org/2010/05/sintel/poster.png">
        <source id='mp4'
            src="http://media.w3.org/2010/05/sintel/trailer.mp4"
            type='video/mp4'>
        <source id='webm'
            src="http://media.w3.org/2010/05/sintel/trailer.webm"
            type='video/webm'>
        <source id='ogv'
            src="http://media.w3.org/2010/05/sintel/trailer.ogv"
            type='video/ogg'>
        <p>Your user agent does not support the HTML5 Video element.</p>

```

```

</video>
<div>
    <input class="left" type='button' onclick='feedback()'
value='give feedback!'/>
    <div class="right" id="feedback">Click Rate: 0.0</div>
    <div class="right" id="flowid">Flow ID: undefined</div>
    <div class="right" id="status">Monitoring status: UNKNOWN
(not started yet)</div>
</div>
</div>
</body>
</html>

```

2.3.4.2 **Basic Web Application : GeoLocation**

This provides an example of how to use the FI-WARE CDI Geolocation API



```

Geolocation is supported
Timestamp: 30 April 2013 15:19:47
Latitude: 53.3748887
Longitude: -6.5253143
Altitude: 0
Lat Long - Accuracy: 70
Altitude - Accuracy: 0
Heading:
Speed:

```

This example shows how the Geolocation API works. It uses CDI to obtain the geographic location of the device, then plots this on a Google Map. It provides full diagnostic information, displaying all the information returned to it by CDI. The following code can be run on top the local web Browser available in your device, and relies on a running PZP instance equipped with the FI-WARE QoE

API Handler module. The following html file will be named as `cdi-geolocation-helloworld.html`.

### *Including the Libraries*

Firstly we must include the correct libraries. In this example application we're also using the jQuery library, but JQuery is not compulsory. It is possible to write this application without jQuery, the author has used it out of convenience. This example uses the Google Map tool to plot the devices current location and therefore also include the Google Map's API.

```
<html>
  <meta name="viewport" content="initial-scale = 1.0,maximum-scale =
1.0" />
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script type="text/javascript" src="jquery-1.9.1.js"></script>
    <script type="text/javascript" src="webinos.js"></script>
    <script type="text/javascript" src="cdi.js"></script>
    <script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=true"></scrip
t>
    <!-- ... -->
  </head>
```

### *Library Initialization*

Before using the CDI API it is necessary to wait for it to Initialize. This is achieved by waiting for the "onCDIReady" event.

```
window.addEventListener("onCDIReady", function(evt) {
  /* ... */
}, false);
```

### *CDI API Discovery*

Once initialized it is necessary to confirm that the platform the application is running on supports geolocation. This is done by checking the `$cdi.sensors.availability.geolocation` boolean value. True indicates that the geolocation functional block is present.

```
window.addEventListener("onCDIReady", function(evt) {
  if ( $cdi.sensors.availability.geolocation ) {
    $("#geo-support").html("is supported");
    startGeoLocationWatching();
  } else {
    /** geolocation is not available **/
  }
}, false);
```



Once the functional block is present the code invokes the `startGeoLocationWatching()` function.

#### *Using the CDI API (Geolocation)*

The `startGeoLocationWatching()` contains all the code which uses the geolocation API. This functional initially sets up the Google Map object before calling the CDI Geolocation functionality.

```
function startGeoLocationWatching() {
    // init the Google map object here.
    var mapOptions = {
        zoom: 12,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new
google.maps.Map(document.getElementById('googlemap'), mapOptions);

    //request the current position

    $cdi.sensors.geolocation.getCurrentPosition(function(position) {
        geolocationPositionCallback(position);
        geolocationWatchId =
    $cdi.sensors.geolocation.watchPosition(geolocationPositionCallback,
    geolocationPositionCallbackError,
        {
            enableHighAccuracy: true,
            timeout: 60000,
            maximumAge: 60000
        });
    }, geolocationPositionCallbackError,
    {
        enableHighAccuracy: true,
        timeout: 60000,
        maximumAge: 60000
    });
};
```

The geolocation functional block is available on the `$cdi.sensors.geolocation` object. The code above requests the current position, if this request completes successfully the lambda function is invoked and map is updated by calling the application function `geolocationPositionCallback` and passing the `position` object supplied by the CDI API. After this the code calls `watchPosition` to register for any changes to the devices location. From this point on every time the devices position changes `geolocationPositionCallback` will be invoked.

### Handling Geolocation Data

The `geolocationPositionCallback` function deals with the geolocation data when it is supplied by CDI. It received the `position` object as supplied by the CDI API. The `position` object contains more than just the latitude and longitude information. It can also include information on speed, if the device is traveling, accuracy information and altitude. The function uses some jQuery calls to update fields on the page page with all the information returned.

```
function geolocationPositionCallback(position) {
    $("#lat").html(position.coords.latitude);
    $("#long").html(position.coords.longitude);
    $("#alt").html(position.coords.altitude);
    $("#ll-accuracy").html(position.coords.accuracy);
    $("#alt-accuracy").html(position.coords.altitudeAccuracy);
    $("#heading").html(position.coords.heading);
    $("#speed").html(position.coords.speed);

    var d = new Date(position.timestamp);
    var dateString = d.toLocaleString();
    $("#timestamp").html(dateString);

    var pos = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
    var marker = new google.maps.Marker({
        position: pos,
        map: map,
        title: 'Hello World!'
    });

    map.setCenter(pos);
};
```

After updating the fields the function then updates the Google map item, by supplying it with the devices current latitude and longitude and updated the marker position.

For more information on the GeoLocation API please see the [CDI Sensor Open Specifications](#).

## 3 Cloud Edge - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

### 3.1 Introduction

#### 3.1.1 Goal of this document

##### 3.1.1.1 *What will be explained*

The goal of this document is to give enough information for programmers to develop and test Virtual Machines on Cloud Proxies HW boxes.

##### 3.1.1.2 *What will NOT be explained*

This document expects programmers to fully understand Linux development as well as application visualization techniques (LxC).

It is also expected that a full cross development environment is installed by the programmers on a dedicated PC.

No User's Guide is provided, as the use of the Cloud Proxy by end users is expected to be described by the developers of applications and services running on it.

##### 3.1.1.3 *Examples*

Several examples of working VMs will be used in this document. These VMs are also provided as working live examples for the programmers.

It is recommended that users are first trying to have these examples running on their Cloud Proxy before trying to develop new ones. It is also recommended to start from the one of the provided example VMs as a basis for developing more complex applications.

### 3.2 Programmers Guide

#### 3.2.1 Create a CloudEdge Virtual Machine image for STB (HTTP web server)

A CloudEdge Virtual Machine image is composed of :

- a manifest file
- a rootfs

For instance, a CloudEdge VM image that provides a HTTP web server can be created :

a) extract the common rootfs template (rootfs-container) used by VMs :

```
$ mkdir rootfs-container
$ tar xzf rootfs-container.tar.gz -C rootfs-container
```

b) build the lighttpd open source webserver using the toolchain (cross-compilation) :

```
$ tar xjf lighttpd-1.4.28.tar.bz2
$ cd lighttpd-1.4.28
$ ./configure --build=i686-pc-linux-gnu --host=i686-cm-linux --without-
pcre --without-zlib --without-bzip2 --prefix=/usr --
libdir=/usr/lib/lighttpd
$ make
```

c) integrate lighttpd in rootfs-container :

```
$ install -d ../rootfs-container/{usr/sbin,usr/lib/lighttpd,var/www}
$ install
src/.libs/{mod_dirlisting.so,mod_indexfile.so,mod_staticfile.so}
../rootfs-container/usr/lib/lighttpd
$ install src/lighttpd ../rootfs-container/usr/sbin/lighttpd
```

d) create lighttpd.conf configuration file in rootfs-container/etc :

```
server.document-root = "/var/www"
index-file.names = ( "index.html" )
mime.type.assign = ( ".html" => "text/html" )
```

and add execute permission on it :

```
$ chmod +x rootfs-container/etc/lighttpd.conf
```

e) create S10lighttpd init script in rootfs-container/etc/init.d (in order to automatically run the webserver when the VM starts)

```
#!/bin/sh lighttpd -f /etc/lighttpd.conf
```

f) create an index.html HTML test page in rootfs-container/var/www :

```
<HTML>
  <HEAD><TITLE>index.html</TITLE></HEAD>
  <BODY>
    <H 1> Lighttpd Sample Home Page </H1>
  </BODY>
</HTML>
```

g) package the Virtual Machine rootfs in FTP server directory located on PC :

```
$ cd rootfs-container
$ tar czf <FTP server directory location>/rootfs-lighttpd.tar.gz *
```

h) create an UUID (uuidgen command) and the manifest.xml file associated to the VM :

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

```
uuid="UUID generated by uuidgen"  
label="lighttpd"  
description="web server"  
rootfs="rootfs-lighttpd.tar.gz">  
</manifest>
```

i) package manifest file in the FTP server directory located on PC :

```
$ tar cf <FTP server directory location>/cloudedge-lighttpd-vm.tar  
manifest.xml
```

### 3.2.2 Run a CloudEdge Virtual Machine image on STB (HTTP web server)

a) Download the CloudEdge web server VM image on STB :

```
$ curl --user smithj:secret http://<STB IP address>:8080/cloudedge/images  
-d '{"image" : {"imageRef" : "ftp://<PC IP address>/cloudedge-lighttpd-vm.tar"}}'
```

b) Create a VM instance of this CloudEdge web server VM image

```
$ curl --user smithj:secret http://<STB IP address>:8080/cloudedge/instances -d '{"instance" : {"imageRef" : 1}}'
```

c) Start the VM instance :

```
$ curl --user smithj:secret http://<STB IP address>:8080/cloudedge/instances/1 -d '{"start" : null}'
```

On PC, test this STB web server by connecting to <http://<STB IP address>> url in a web browser

### 3.2.3 Run others CloudEdge Virtual Machines on STB

```
tbd ...
```

## 4 Network Information and Control - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

- [NetIC GE - OFNIC Uniroma - User and Programmers Guide](#)

## 5 NetIC GE - OFNIC Uniroma - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

### 5.1 Introduction

OFNIC Uniroma provides a programmable interface to retrieve information and control Openflow Networks. The interface is based on the RESTful paradigm and is powered by a Web Server developed in Python programming language. OFNIC Uniroma relies in the Openflow protocol in order to abstract and virtualize forwarding capabilities of the Openflow Network, but also SNMP protocol to retrieve more accurate statistics. This GE is an extension to the NOX Openflow controller. OFNIC Uniroma is composed of many modules which communicate with each other with events. It is designed with an event-driven paradigm also to manage information that comes from the network. The OFNIC Uniroma RESTful interface is an instance of the NetIC API (NetIC Open Specifications can be found here TBC). Goal of this document is to provide a useful guide for developers which might build new applications based on the NetIC API, or a guide for users that might utilize the provided GUI application. Specific sections are dedicated to developer tools intended to help them during development, test and deployment.

### 5.2 User Guide

The NetIC RESTful API reference offers a [\[1\]](#) page, which describes how to use this API in details. It includes a set of commands, which can be used to manipulate and instantiate network resources physical or virtual. Since NetIC RESTful API is based on the HTTP protocol there are a rich variety of tools, which can be used to access the OFNIC Uniroma GE interface. The following sections will go into more detail on how to use and develop against NetIC RESTful API.

### 5.3 Programmer Guide

In this section it is supposed that the GE is up and running. See Installation and Administration guide for this.

#### 5.3.1 Navigation with browser

The root path of the Web Server is:

```
https://127.0.0.1/netic.v1/doc
```

This page shows all commands that can be sent to the REST API of OFNIC Uniroma. Note that not all the methods and commands might not be supported directly from the browser, that is because some browser does not support natively the `DELETE` Http command.

Regarding `POST` command the parameters of the request might be:

- encoded in the url

```
field1=value1&field2=value2&field3=value3...
```

- formatted as JSON objects in the body of the Http Request.

```
{field1 : value1, field2 : value2, field3 : value3}
```

#### 5.3.1.1 *Examples*

In this example the OFNIC Uniroma GE is connected to an Openflow network composed of three Openflow nodes. The nodes run Openvswitch for implement the Openflow protocol and SNMPd and SNMP Sub-Agent to collect statistics. The OFNIC Uniroma GE is located at the same machine from which the browser is used. Using the browser, type in the location bar the following URL string and then press Enter.

```
https://127.0.0.1/netic.v1/OFNIC/synchronize/network
```

The result displayed, should be the list of nodes present in the network:

```
{
  "resultCode": 0,
  "displayError": "No error",
  "result": {
    "Paths": [],
    "Nodes": [
      1,
      2,
      3
    ]
  }
}
```

In order to retrieve information about the configuration of node with ID equal to 2, the following string should be typed in the URL container:

```
https://127.0.0.1/netic.v1/OFNIC/synchronize/network/node/2
```

The following JSON response should appear in the Web browser:

```
{
  "resultCode": 0,
  "displayError": "No error",
  "result": {
    "Port_Names": [
      "eth3",
      "eth2",
      "br0",
      "eth1"
    ],
    "Port_Index": [
```



```

        3,
        2,
        65534,
        1
    ],
    "Num_Buffers": 256,
    "Actions": 4095,
    "Num_Tables": 255
}
}

```

The same operation can be repeated to get information about port 1 of node 2, with the following URL:

<https://localhost:2222/netic.v1/OFNIC/synchronize/network/node/2/port/1>

,the following result is expected:

```

{
  "resultCode": 0,
  "displayError": "No error",
  "result": {
    "Active": true,
    "Config": 0,
    "State": 0,
    "Speed": 0,
    "links": [
      0
    ]
  }
}

```

To retrieve statistics about a port of a certain node (for example port 1 of node 2) the following URL might should be used:

<https://localhost:2222/netic.v1/OFNIC/statistics/node/2/port/1>

and the results that appears in the browser shows the transmitted and received bytes of the port:

```

{
  "resultCode": 0,
  "displayError": "No error",
  "result": {
    "Tx_bytes": 34,

```

```

        "Rx_bytes": 20
    }
}
    
```

As one can note from the examples all responses body are in JSON format and three fields are always present:

- resultCode: 0 for no errors, any other number for the occurring error.
- displayError: a string that displays the type of error
- result: the result of the request.

### 5.3.2 OFNIC Uniroma GUI (*draft version*)

The OFNIC Uniroma GE is released with a GUI feature (to be released definitively in June 2013). The NetIC API users might utilize the GUI to navigate the exposed RESTful webservices. The GUI has been developed in Javascript language, and uses the FI-WARE site template. The web GUI comes embedded in the OFNIC source code, to access it from a browser navigate to:

```

http://localhost/static/0/nox/webapps/webserver/dummy/index.html
    
```

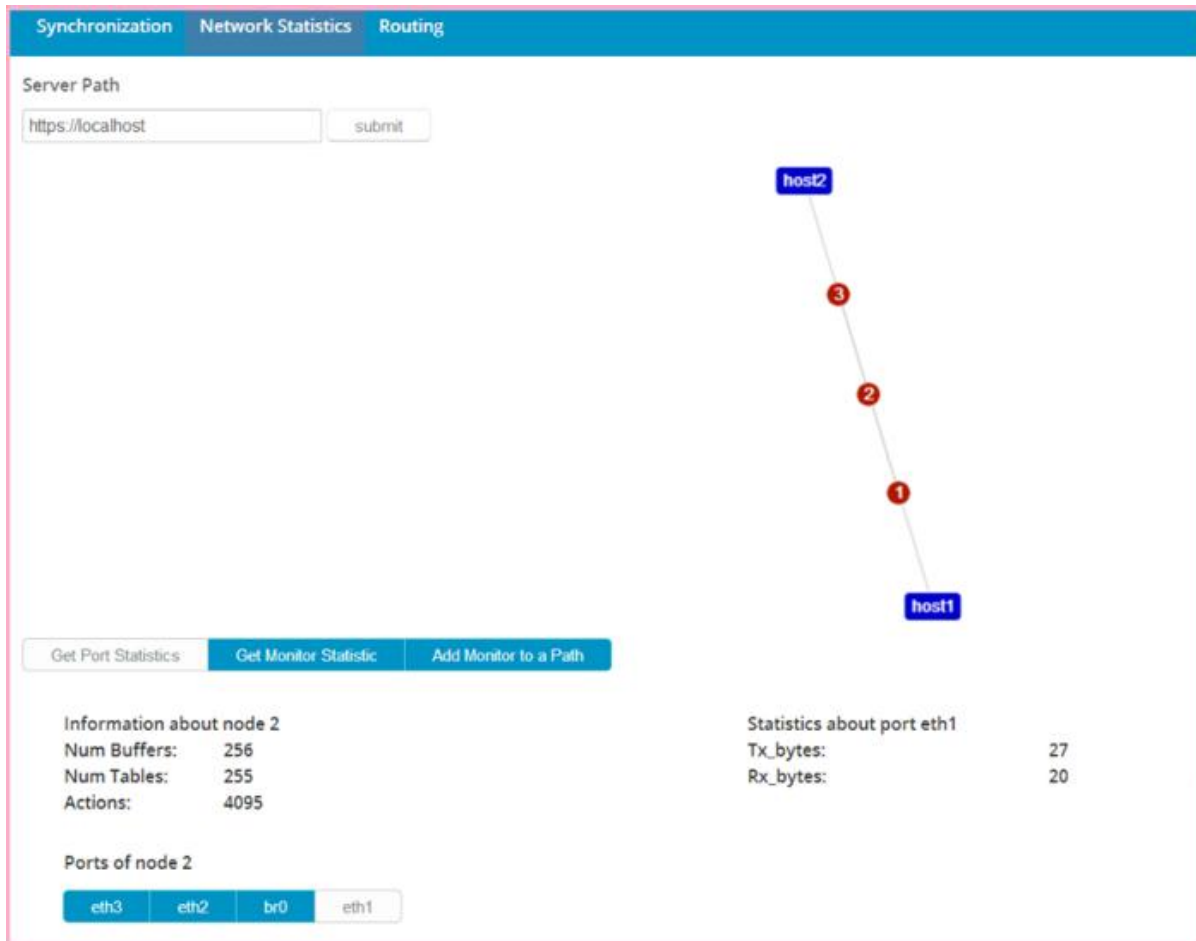
The GUI page that appears is shown in the figure below.



In the figure above one can note that the GUI is composed of three navigation tabs: *Synchronization*, *Statistics*, and *Routing*. The *Synchronization* tab shows the network topology with an interactive diagram. Network nodes are clickable and additional information is shown if a node

is selected. For example the number of Actions or Ports. Moreover the Displayed ports are clickable and show to which other network entity is the selected node connected to.

The Statistics navigation tab, concerns about the presentation of network statistics collected by the OFNIC Uniroma GE. The dynamic network topology is present so one can select nodes he might wish to monitor from the graph. The Statistics tab is shown in the figure below.



### 5.3.3 Apache HttpClient

Programmers that might be interested in developing applications that rely in the NetIC RESTful API might consider in using well-known and already established REST clients and REST client libraries. Apache Software Foundation provides a powerful package HttpClient. HttpClient is an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations. It is simple to use, so even not very experienced programmers can build simple application in a few minutes. Some code snippets are reported below.

```
HttpClient client = new HttpClient();
GetMethod request = new GetMethod("https://localhost/netic.v1/doc");
try {
    HttpResponse response = client.execute(request);
    HttpEntity entity = response.getEntity();
}
```

```

    if (entity != null) {
        BufferedReader rd = new BufferedReader
            (new InputStreamReader(response.getEntity().getContent()));
        String line = "";
        while ((line = rd.readLine()) != null) {
            textView.append(line);
        }
    }
} catch (ClientProtocolException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

```

OFNIC Uniroma GE is released, accompanied by a Java application based on Apache HttpClient that was designed in the development and testing phase.

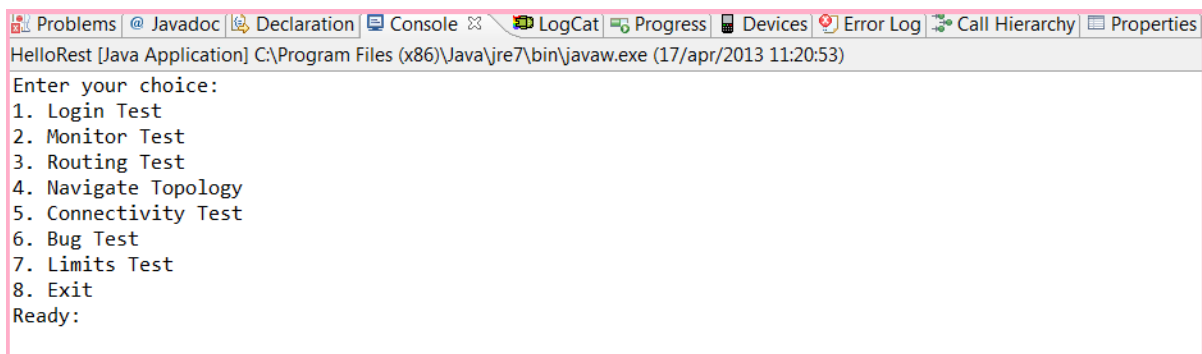
The source code of the application named "NetIC\_API\_Tester can be downloaded from the FI-WARE svn repository:

```

svn co https://forge.fi-ware.eu/scmrepos/svn/fiware/trunk/FI-WARE/I2ND/NetIC/OFNIC/NetIC\_API\_Tester netic_api_tester

```

The application can be executed on Linux environment by running the script `test.sh` located inside the `src` folder. A list of test appears that can be executed by putting in the corresponding number.



```

HelloRest [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (17/apr/2013 11:20:53)
Enter your choice:
1. Login Test
2. Monitor Test
3. Routing Test
4. Navigate Topology
5. Connectivity Test
6. Bug Test
7. Limits Test
8. Exit
Ready:

```

The `Netic_API_Tester` can be imported also in Eclipse[ECL] as a normal project. To do this, after the source code has been checked out locally, in Eclipse IDE go to File->Import... In the dialog box that appears under General choose Existing Project into Workspace... Following the wizard one should have imported the project in the IDE and can manage it easily.

#### 5.3.4 How to extend OFNIC Uniroma GE

OFNIC Uniroma GE is an extension to the NOX Openflow Controller, so the guide to extend NOX can be utilized to extend also OFNIC. The guide is located in this page:

<https://github.com/noxrepo/nox-classic/wiki/Developing-in-NOX>

## 6 Service Capability Connectivity and Control - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

- [S3C GE - OpenEPC - User and Programmers Guide](#)
- [S3C GE - SMS+MMSEnablers - User and Programmers Guide](#)

Appendix:

- [S3C GE Support - S3C API Proxy - User and Programmers Guide](#)

## 7 S3C GE - OpenEPC - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

### 7.1 Introduction

The EPC-OTT API enables the application developers to use a RESTful interface within their application to provide their indications towards the operator core network. The following section describes how the application developers can influence the communication through the operator network.

The following features are available from the core network perspective:

- Reservation of resources for the specific communication sessions and for each of the session's data flows
- Modification of the previously reserved resources
- Termination of the previously reserved resources
- Event notifications in case the status of the connectivity of the mobile devices changes

These features are provided through the Rx Diameter interface. The EPC-OTT API is providing a translation between the Diameter interface towards the operator core network and the RESTful interface towards the applications.

Additionally to these features, the EPC-OTT API provides through the RESTful API the applications with information on the access networks on the vicinity of the mobile devices and a means to transmit an indication on which access network the communication of the application should continue on. As multiple applications run in parallel and as the final decision on which access network is most suitable to support the device communication is pertaining to the operator, the indication from the application may be ignored.

#### 7.1.1.1 *Restrictions*

The EPC-OTT API component is delivered in a testbed demonstration version and is bound to a limited functionality.

Additionally, the operator core network functionality was not designed to be exposed even to applications for which a business relationship with the operator exists. Therefore, the EPC-OTT exposure of the operator core network is restricted also by the exposure features offered by the underlying operator infrastructure.

The following limitations are considered:

- The QoS resource reservations, modifications, and termination are bound to the International Mobile Subscriber Identifier (IMSI) and not to other possible identities
- The access network selection allows only for access network type selection and not for the selection of the specific access networks in the vicinity of the device. A finer granularity requires a higher exposure of the internals of the operator network, which may pose a security threat.

## 7.2 User Guide

The EPC-OTT API component, handling the translation between HTTP/REST messages and typical operator core network messages such as Diameter was developed in parallel with a test client application.

The test client application is written in Java and able to transmit the possible requests and to receive the appropriate responses from the EPC-OTT API>

The test client provides an initial implementation example, which can be downloaded together with the EPC-OTT API Component providing the developers a first hands-on example on how the API can be used.

The Test Client application is developed in Java. Thus, the following code snippets are written in this programming language.

A main class containing the QoS related information for a specific data flow is `QoSFeatureProperties`

```
public class QoSFeatureProperties implements Serializable
{
    private static final long serialVersionUID = 12343L;
    protected TimeMetric duration;
    protected String upStreamSpeedRate;
    protected String downStreamSpeedRate;
    protected List<NameValuePair> otherProperties;

    /* Functions to set and get the different QoS features */
}
```

For each session initiation or modification an object of the `QoSFeaturesProperties` is filled with the specific information related to the QoS parameters of each data flow.

The array of `otherProperties` may include the description of the specific data flow including source and destination IP addresses and ports, protocol and direction of communication.

When such information is missing, it is assumed that a wild-card is used. For example if the destination port is missing, then the QoS rule will be applied for all the data packets which match the rest of the filtering criteria.

A `WebResource` object (`com.sun.jersey.api.client.WebResource`) enables the encapsulation of the String represented by the information within the `QoSFeaturesProperties` class within an HTTP request which is forwarded to the URI of the EPC-OTT API component, enabling an easy and elegant means to realize an HTTP interface within Java based software.



## 7.3 Programmer Guide

The EPC-OTT API could be extended with additional functionality by using the same basic Jetty server functionality.

Additionally, in order to be able to translate the specific RESTful interface to the specific operator core network functionality, the specific operator core network stacks have to be used such as Diameter and other proprietary communication protocol.

The only specific properties of the network which can be manipulated by the different applications are the ones that could be exposed by the underlying network infrastructure.

# 8 S3C GE - SMS+MMSEnablers - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

## 8.1 Introduction

The SMS/MMS enabler allows you to easily integrate with SMS GSMA One API and perform end-2-end unitary tests. The following sections provides detailed information on how to integrate SMS sending capabilities into your application. The following features are available:

- Send SMS
- Query SMS delivery status
- Start subscription for SMS delivery status
- Stop subscription for SMS delivery status
- Receive SMS
- Start subscription for SMS receipt
- Stop subscription for SMS receipt

### 8.1.1 Restrictions

The SMS/MMS enabler is delivered in a Demo version which is bounded by some simple rules:

- Limitation of the network: one can send SMSs to the Orange network only to Orange France, Orange UK, Mobistar and Orange Romania.
- Limitation of the number of SMS sent

### 8.1.2 Countries, Short Codes and sender addresses

The following short codes can be used as senderAddress depending on the country code the recipient (i.e. MSISDN) is belonging too (e.g. **336xxxxxxx** -> Orange France).

You can also provide a MSISDN of those countries as senderAddress, but in such a case this account will be charged with the cost of the sent SMS MT message.

Country	Perimeter	Short code to use as sender
France	OFR only	21238
United Kingdom	OUK only	91255

Belgium	Mobistar only	10403
Romania	OROnly	587

**NOTE 1:** Like you are using real production platform, it override received 'senderAddress' value received from your application to be in line with requirements. Keep in mind that if your application goes on production (ie out of the fiware testbed), you'll have to properly fill the 'senderAddress' value with you own short code(s).

**NOTE 2:** What ever the receipt country is, if you don't want to "really" send a SMS (i.e do not send SMS to cell phone network and have a SMS delivery on a mobile phone) but you just want to have an answer "like an answer from the enabler should be" use 99999 as shortCode.

Returned value will look like a valid answer from the enabler, but SMS-ID will a fixed value: -1

## 8.2 User guide

### 8.2.1 1/Pre-requisites

This API allows you to send text/plain SMS MT message using an HTTP GET request with a few number of parameters.

Please consult the [SMS/MMS API description \(interface level\)](#) section for further information about the '/messaging/sms/outbox' resource URI, supported HTTP verb (GET), mandatory/optional input parameters, supported representation formats, etc.

### 8.2.2 2/Sending your first SMS message

The SMS MT GSMA One API specification is a restful implementation, thus working with simple URLs, dialoging with JSON data and using http verbs GET, POST...

The request

```
http://ServerName/oneapi-sms/outbound/sender/requests
```

allows you to send an SMS from *sender* short code or identifier. The other information is embedded in the body of the request in a JSON formatted text. The following content:

```
{
  "outboundSMSMessageRequest":
  {
    "address": ["33600000000"],
    "senderAddress": "33611111111",
    "outboundSMSTextMessage": {"message": "Hello World"},
    "senderName": "33611111111",
    "receiptRequest": {"notifyURL": "http://myexample/notification"}
  }
}
```

The consequence is that the mobile "3360000000" with receive an SMS "Hello world", from the 336111111111. The notification url should be a reachable adress and will get the following answer:

```
{
  "outboundSMSMessageRequest": {
    "address": ["3360000000"],
    "senderAddress": "336111111111",
    "outboundSMSTextMessage": {
      "message": "Hello World"
    },
    "senderName": "336111111111",
    "receiptRequest": {
      "notifyURL": "http://myexample/notification"
    },
    "resourceURL": "http://ServerName/oneapi-
sms/outbound/sender/requests/1366381061048",
    "deliveryInfoList": {
      "resourceURL": "http://ServerName/oneapi-
sms/outbound/sender/requests/1366381061048/deliveryInfos",
      "deliveryInfo": [{
        "address": "336111111111",
        "deliveryStatus": "MessageWaiting"
      }]
    }
  }
}
```

Note that the delivery status "MessageWaiting" informs you that your message has not been delivered yet. The notification, after delivery will be sent to <http://myexample/notification>

### 8.3 Programmer guide

The SMS/MMS enabler follows the API specifications of the GSMA OneAPI. You will find the complete developer guide at this address: <http://www.gsma.com/oneapi/faq/implementation-guides>

## 9 S3C GE Support - S3C API Proxy - User and Programmers Guide

You can find the content of this chapter as well in the [wiki](#) of fi-ware.

### 9.1 Introduction

The S3C API Proxy is just a supporting function to enable a single point of contact to utilize the HTTP/REST based Features of S3C.

### 9.2 User guide

To use the functionality of the S3C API Proxy, there is only a need for reviewing the following User and Programmers Guides:

- [S3C GE - OpenEPC - User and Programmers Guide](#)
- [S3C GE - SMS+MMSEnablers - User and Programmers Guide](#)

### 9.3 Programmer guide

For editing the mapping functionality of the API Proxy, only one specific configuration file needs to be changed:

```
/etc/apache2/mods-available/proxy.conf
```

In this file the ProxyRequests key has to be set to "on".

To add a special feature resource, two additional lines between <IfModule>...</IfModule> need to be set:

```
ProxyPass /<proxyresourcename>/
http://<featureip>:<featureport>/<additionalresource>/
ProxyPassReverse /<proxyresourcename>/
http://<featureip>:<featureport>/<additionalresource>/
```

An example looks like:

```
ProxyPass /openepc/ http://192.168.254.40:80/
ProxyPassReverse /openepc/ http://192.168.254.40:80/
```

This means, that the feature OpenEPC is available at the following address of the proxy:

```
http://<proxyaddress>:<port>/openepc/
```

All functionalities of the remote targeted feature are useable over the proxy.