



**Business  
Services**

# **M2M API for Java**

## **SDK Manual**

# M2M API for Java: SDK Manual

Copyright © 2010 Orange

## Revision History

Revision 1.0.6-000	2010/03/19
• First release.	

# Table of Contents

Preface .....	iv
1. Overview .....	1
1. Features .....	1
2. Concepts .....	1
2. Prerequisites .....	3
1. What you need to consume M2M API web service .....	3
2. Manage certificates .....	4
3. Java environment .....	5
4. Proxy and firewall settings .....	5
3. Install and Setup .....	6
1. Install M2M SDK in Java .....	6
2. Configure SDK .....	6
4. The M2M SDK .....	9
1. Common .....	10
2. Malima .....	12
3. Generated .....	15
4. SDK configuration .....	15
5. Logging .....	16
6. First project .....	20
5. Methods .....	25
1. Basic scenarios .....	25
2. GetSubscriptionStatus method .....	26
3. GetConnectivityDirectory method .....	29
4. SearchInConnectivityDirectory method .....	32
5. UpdateConnectivityDirectory method .....	36
6. SubmitUpdateSimStatus and GetUpdateSimStatus methods .....	41
7. SubmitConsumptionTracking and GetConsumptionTracking methods .....	46
8. GetNetworkStatus method .....	51
9. GetIncidentDiagnostics method .....	55
10. SubmitSessionHistory and GetSessionHistory methods .....	58
A. Error codes .....	63
1. Connectivity .....	69

# Preface

Welcome to the M2M API SDK manual!

This manual is a collection of topics related to develop code using all the advanced features of Orange™ M2M API with our SDK.

Note : M2M API is also known as Malima API

By the end of this manual you will be able to make advanced applications using our API. You will discover the implementation details (architecture and libraries used) of SDK we made for you. Our SDK has been designed to hide all the complexity of the underlying WebService communication and to help leverage the power of Object Oriented Programming.

You will first learn how to setup your machine, then how to configure the SDK and finally how to use every single methods provided by the API.

You will find some code snippets that are ready to go, just copy and paste the code into your favorite IDE!

# Chapter 1. Overview

## 1. Features

The M2M webservice offers a way of managing SIM cards set, during the construction phase, the setting phase, or the operating phase.

M2M API offers the following features:

- Get the data set of a SIM card and refresh it.
- Get SIM information according to a collection of up to 100 line identifiers.
- Search SIM information.
- Update a set of SIM referencials, including devices and machines.
- Update a set of SIM status.
- Request for a set of SIM traffic in a period of time.
- Get the network covering status into an area defined by its latitude and longitude.
- Get the diagnostic analysis of a SIM card, including the network area covering the SIM card.
- Request for a SIM card set of statistics by session in a period of time.

## 2. Concepts

The provided services are web services included in a Software Development Toolkit, that can be inserted in software applications. They allow automating the deployment and the management of a set of machines, embedding SIM cards, and exchanging data with a central server thanks to a mobile connection (GSM / GPRS).

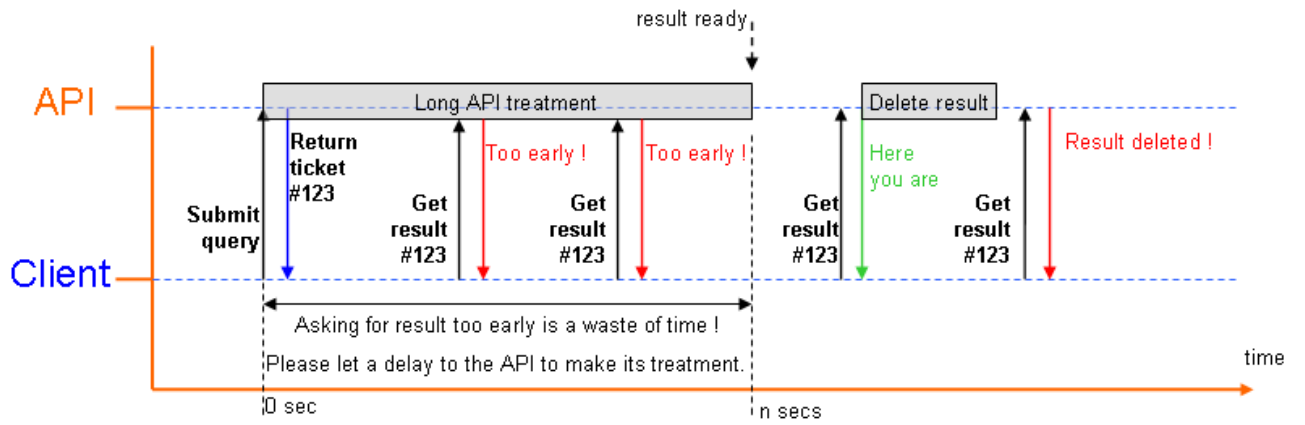
Two kind of actors are concerned by the Orange M2M services :

- The operator of the machines set, who expresses functional demands in relation to his management applications.
- The developers who insert these services in the given application.

### Tip

Please note that some operations are done in asynchronous mode. They need a long time treatment (in case of M2M updates for example). A ticket number is returned when the method is called. Later, this ticket number can be used to get final result. Please use these operations very carefully. As the methods can be long lasting, it is unnecessary to require the operation result using the ticket number too frequently and too early after getting ticket number. A small figure below explain how to query your result.

Figure 1.1. Example of asynchronous sequence call with result get later



# Chapter 2. Prerequisites

Before getting started, please read the information of this document, and follow any installation / download instructions.

In this document you will learn how to get your credentials. Then, we will explain how to have a well configured java environment.

## 1. What you need to consume M2M API web service

To access the M2M API, you first have to get from Orange some credentials. Once you have these items, please keep them, they will be necessary to access the M2M API.

### 1.1. API access credentials needed

This section focuses on how to access the API and consume the webservice.

Developing with the API, you'll need:

- *Service URLs*: the URLs to consume the webservices
- *Access Key* and *Access Key Password* for basic authentication.

### 1.2. M2M API requirements

You will need the relevant end point URL when using the API. You need to use different URLs for different functionalities of your development.

The following table recaps the WSDL endpoint locations depending on the M2M functionality you are working with.

**Table 2.1. Which endpoint should I use?**

Functionality	URL
Subscription Status	<a href="https://iosw-ba.francetelecom.com/MLM/SubscriptionStatus-1">https://iosw-ba.francetelecom.com/MLM/SubscriptionStatus-1</a>
Connectivity Directory	<a href="https://iosw-ba.francetelecom.com/MLM/ConnectivityDirectory-1">https://iosw-ba.francetelecom.com/MLM/ConnectivityDirectory-1</a>
Sim Lifecycle Management	<a href="https://iosw-ba.francetelecom.com/MLM/SimLifecycleManagement-1">https://iosw-ba.francetelecom.com/MLM/SimLifecycleManagement-1</a>
Consumption Tracking	<a href="https://iosw-ba.francetelecom.com/MLM/ConsumptionTracking-1">https://iosw-ba.francetelecom.com/MLM/ConsumptionTracking-1</a>
Network Status	<a href="https://iosw-ba.francetelecom.com/MLM/NetworkStatus-1">https://iosw-ba.francetelecom.com/MLM/NetworkStatus-1</a>
Incident Diagnostics	<a href="https://iosw-ba.francetelecom.com/MLM/IncidentDiagnostics-1">https://iosw-ba.francetelecom.com/MLM/IncidentDiagnostics-1</a>
Session History	<a href="https://iosw-ba.francetelecom.com/MLM/SessionHistory-1">https://iosw-ba.francetelecom.com/MLM/SessionHistory-1</a>

Download and install the server certificate from M2M API on your machine in order to authenticate your application for the Secure Socket Layer (SSL).

## 2. Manage certificates

### 2.1. Secured Communication

HTTPS (Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL) is a Web protocol designed to encrypt and decrypt page requests as well as the pages that are returned by the Web server.

The communication between your machine and the API will be secured by a SSL server certificate over the regular HTTP protocol. Practically this means you will have to go through HTTPS endpoints and you will have to use the SSL certificate we've made available for you. It is named `orangeapi.cer`.

### 2.2. Use a certificate in Java

#### Tip

We have also decided to provide you with the certificates in java format in the SDK so you can skip this section, nevertheless, we encourage you to read it to achieve yourself. It will have a default password value of *changeit*.

Java uses its own file format to store certificates called *keystore* (.jks files) so you will have to use the **keytool** command to interact with this file format.

As the purpose of this document is to create simple API calls we assume you have no preexisting environment that you have to conform to (ie.: web server, ...). The sdk binary archive contains a pre-configured keystore with the needed certificates.

If you want to use your own keystore, you may need to get the certificates on the Orange Partner site (before getting started section), then type the following command for each downloaded certificate.

```
$>keytool -import -alias <your_alias> -file <certificateFile.cer> -keystore
<keystoreFileName.jks> -storepass <password_keystore>
```

To check that your keystore file has correctly been set, you can use the `list` option of the **keytool** command:

```
$>keytool -list -keystore <keystoreFileName.jks> -storepass <password_keystore>
```

If you specify the "password\_keystore", you will have to specify the password in the argument when calling the initialize certificate function in the code. If you do not specify any password, the default password "changeit" is set. The password to the keystore provided in the binary archive is set to its default value.

#### Tip

If you want to have **keytool** command directly available from your shell you may have to configure your executable path to add a reference to java binaries directory.

If you are on a Windows platform you can type the following command:

```
$>set PATH=%PATH%;<path to you java installation>\bin
```

If you are on an Unix platform you can type the following command:

```
$>export PATH=$PATH:<path to you java installation>/bin
```

Then check if it worked by observing the result of the command **keytool**.

### 3. Java environment

We encourage you to use Integrated Development Environment (IDE) which will ease your everyday work with programming. You can download some opensource ones for free like [NetBeans](#) or [Eclipse](#).

### 4. Proxy and firewall settings

In order to use our APIs, you have to check your proxy and firewall settings.

The firewall basically inspects network traffic passing through it. It may deny passage based on how rules have been set by your system administrator. We recommend to first check with the administrator if the port is open and if the URL request won't be blocked. Once the proxy and firewall are well set, we encourage you to test the network with a simple function that takes a few easy-to-build parameters and just returns a status. This will allow you to verify that everything is working fine.

# Chapter 3. Install and Setup

To kick-start your development download the JAVA JDK. The reference manual (generated by javadoc) is also available in the ZIP of the SDK.

## 1. Install M2M SDK in Java

The SDK is composed of a main library `malima-<version>.jar` and a set of dependency libraries (`.jar`), these libraries are all mandatory. What you have to do now is rather simple if you followed previous document sections. Just put all the libraries, the configuration file named `malima.properties` and `orangeapi.jks` within your project.

Here is a typical project structure overview:

```
<project>
- src
  - malima.properties
  - orangeapi.jks
  - <your java sources>
- lib
  - malima-<version>.jar
  - <Malima SDK java dependency libraries>
  - <your java libraries if you have any>
```

### Tip

We encourage you to use Integrated Development Environment (IDE) which will ease your everyday work with programming. You can download some opensource ones for free like [NetBeans](#) or [Eclipse](#).

## 2. Configure SDK

The SDK is shipped with a configuration file that stores all settings required to access the API. You can also to configure the API through coding.

- the web proxy settings: `proxyHost`, `proxyPort`, `proxyUsername` and `proxyPassword`,
- the service Urls,
- the access credentials: `accessKey` and `accessKeyPassword`.

This configuration file is named `malima.properties` and has to be located at the classpath root of your application.

**Example 3.1. Example of malima.properties file**

```

#-----
#      (1) Configuration service
#-----
# http proxy (optional)
proxyHost = yourProxyHost
proxyPort = yourProxyPort
proxyUsername =
proxyPassword =

# SSL: trust file or public key (server authentication)
# by giving an absolute path
# [Windows]      : C:\\yourKeyStoreFile.jks
# [Linux, Unix]  : /yourKeyStoreFile.jks
#
# by giving the relative path of the classpath root (for advanced user)
# [Windows]      : ..\\yourFolder\\yourKeyStoreFile.jks
# [Linux, Unix]  : ../yourFolder/yourKeyStoreFile.jks

# keystore path file (mandatory)
keyStoreFile = C:\\yourKeyStoreFile.jks

# keystore password (optional)
keyStorePassword = yourKeystorePassword

#-----
#      (2) Configuration API (mandatory)
#-----
# Credentials #
accessKey=your_access_key
accessKeyPassword=your_access_key_password

# ConnectivityDirectory Configuration #
connectivityDirectoryUrl= your_connectivity_directory_url

# ConsumptionTracking Configuration #
consumptionTrackingUrl= your_consumption_tracking_url

# IncidentDiagnostics Configuration #
incidentDiagnosticsUrl= your_incident_diagnostics_url

# NetworkStatus Configuration #
networkStatusUrl= your_network_status_url

# SessionHistory Configuration #
sessionHistoryUrl= your_session_history_url

# SimLifecycleManagement Configuration #
simLifecycleManagementUrl= your_sim_lifecycle_management_url

# SubscriptionStatus Configuration #
subscriptionStatusUrl= your_subscription_status_url

```

**Tip**

keyStoreFile property expects a path value so do not forget that the format of the path is depending on the OS you are running.

## Tip

Please escape the back slash character '\' in attributes of the property file. For example, characters of the password 4fg!ws\ must be escaped as follow: 4fg!ws\\

This configuration file is not mandatory (you can set all required parameters programmatically), but we encourage you to use it as it greatly eases the maintainability of your application.

# Chapter 4. The M2M SDK

Hereafter is described a global view of the M2M SDK architecture. For a more precise functional view, please refer to the functional user document.

Basically, the M2M SDK is based on Java 5 framework and is composed of:

- Common: this library contains reusable classes in the Open Developer Network context.
- Malima: this library contains
  - the `MalimaService` with static access to the Malima API webmethods.
  - the `MalimaUser` class, encapsulating authentication data.
  - the Business Model package, a business model that lets the developer get quickly involved in the M2M API
  - the generated package, a communication layer with SOAP client and types definitions.

**Figure 4.1. Diagram of the global architecture of the M2M SDK**

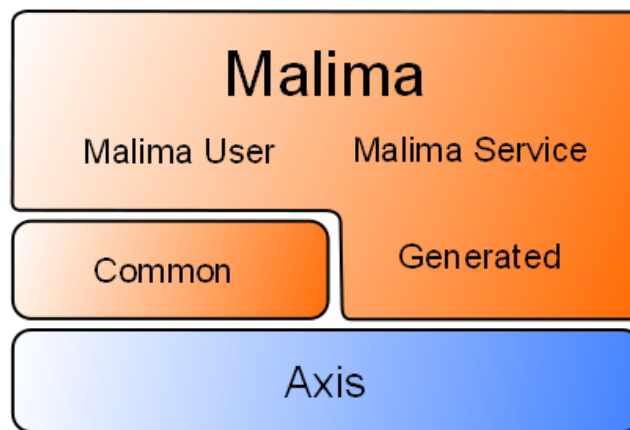
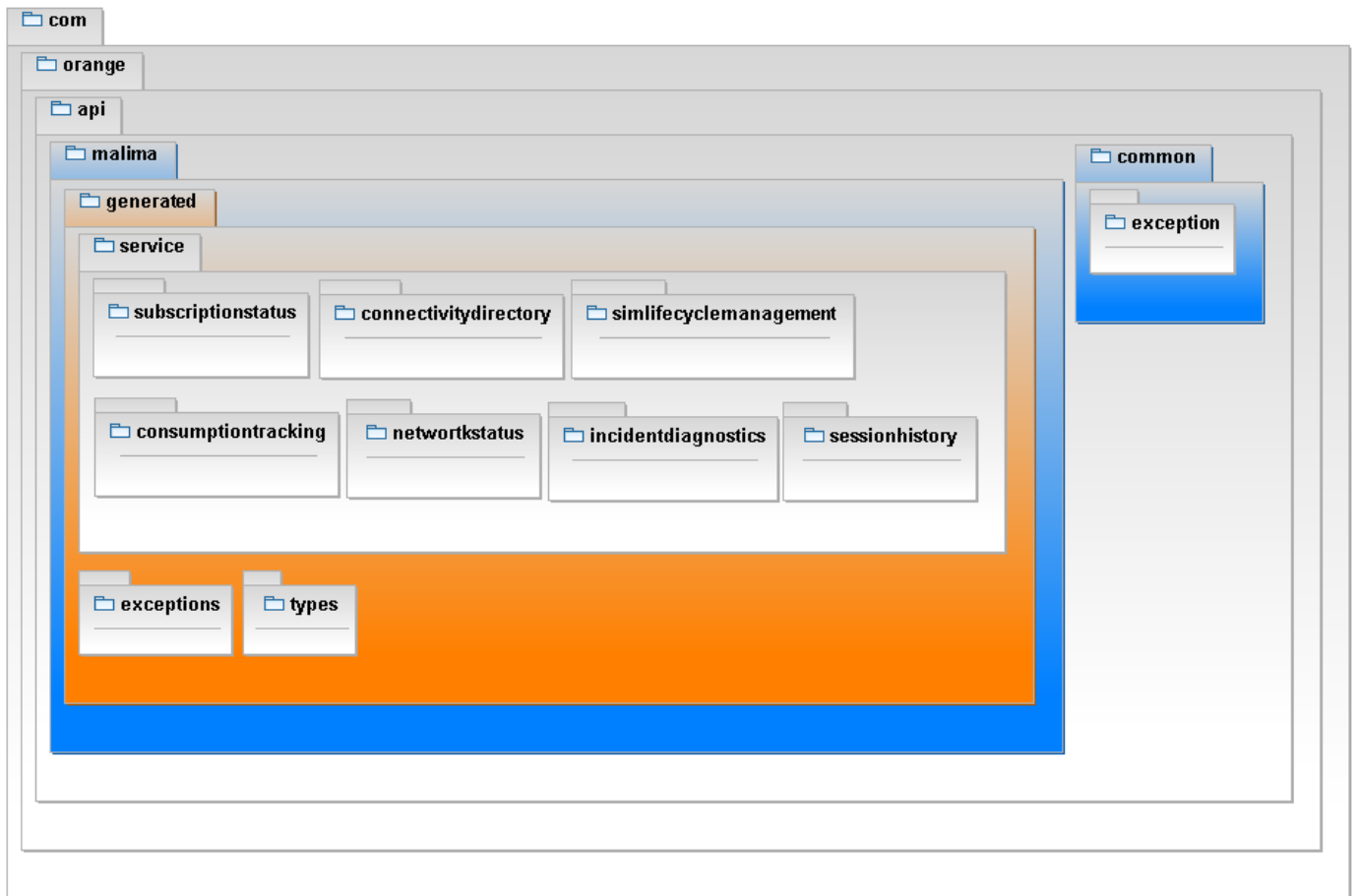


Figure 4.2. UML model diagram of Malima model



## 1. Common

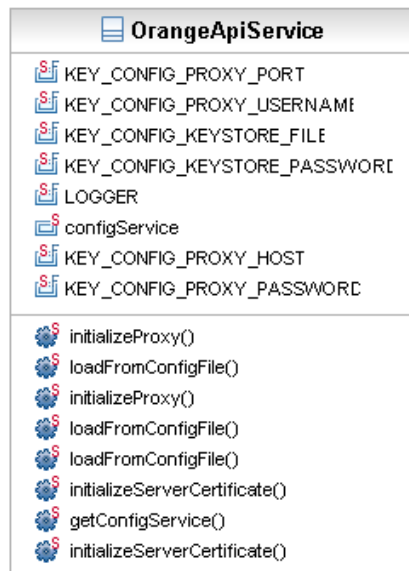
The `com.orange.api.common` package is made of two main classes.

- `OrangeApiUser` class: encapsulates the authentication data required by the Orange APIs.
- `OrangeApiService` class: encapsulates security methods such as proxy configuration or initialization of a server certificate.

**Figure 4.3. OrangeApiUser UML class diagram.**



**Figure 4.4. OrangeApiService UML class diagram.**

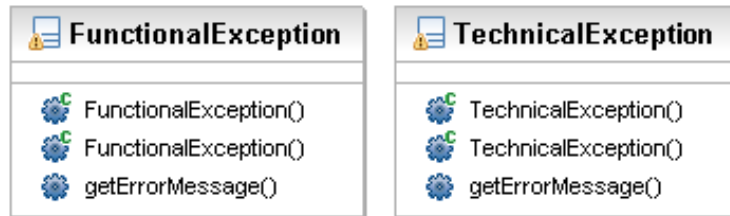


The com.orange.api.common.exception package is made of two main classes.

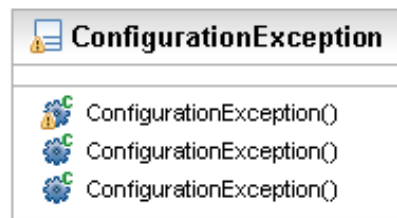
- FunctionalException class: encapsulates an exception that had been raised by the Orange API.

- `TechnicalException` class: encapsulates an exception that had been raised before reaching the Orange API.
- `ConfigurationException` class: encapsulates an exception that had been raised during configuration of the SDK.

**Figure 4.5. Common exception UML class diagram.**



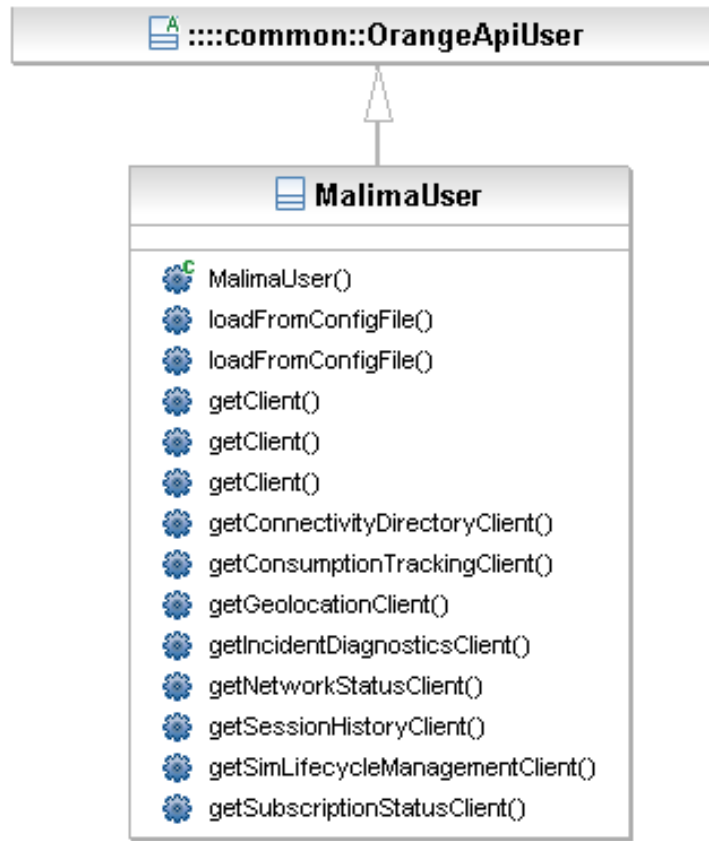
**Figure 4.6. Common configuration exception UML class diagram.**



## 2. Malima

The `com.orange.api.malima` package contains three classes.

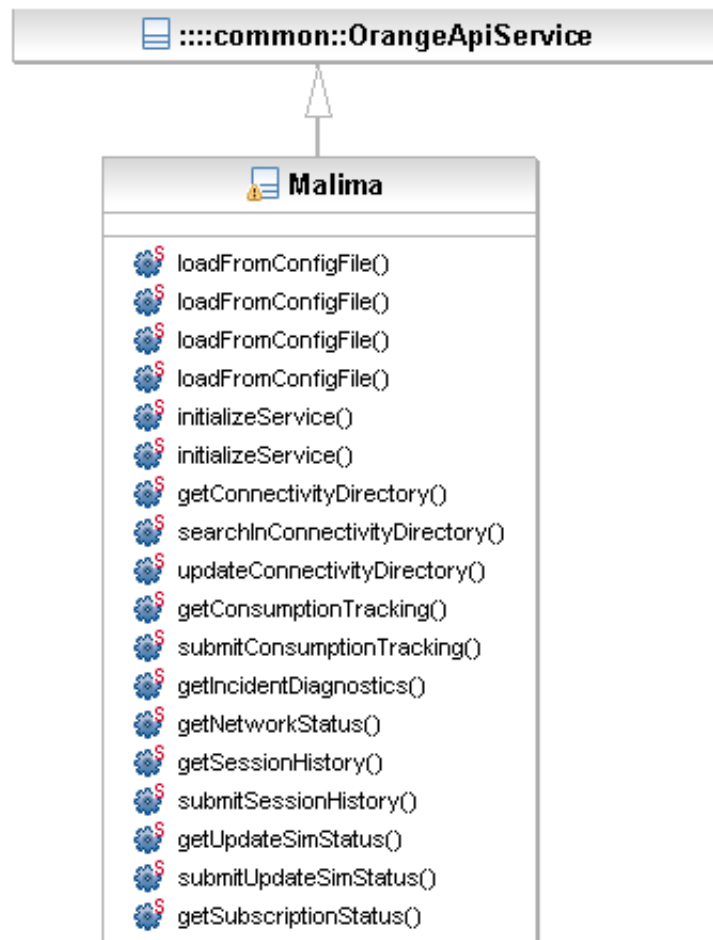
- `MalimaUser`: this class represents a configurator helper for M2M API. It inherits from the `com.orange.api.common.OrangeApiUser`.

**Figure 4.7. MalimaUser UML class diagram.**

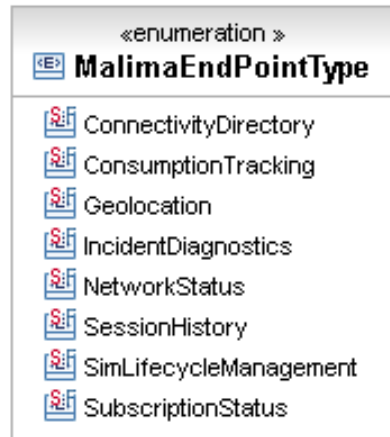
- **Malima:** this class represents a simple entry point to call webmethods of the M2M API. It inherits from the `com.orange.api.common.OrangeApiService`, thus encapsulates static initialization methods. Moreover, it provides business and configuration methods :
  - `getSubscriptionStatus`: Get the data set of a SIM card and refresh it, according to one of its identifiers.
  - `getConnectivityDirectory`: Get SIM card information according to a collection of up to 100 line identifiers.
  - `searchInConnectivityDirectory`: Search SIM card information according to a set of sorted parameters, returned by range.
  - `updateConnectivityDirectory`: Update a set of SIM card referentials, including devices and machines, accordig to SIM cards's identifiers.
  - `submitUpdateSimStatus`: Update a set of SIM cards's statuses, according to their identifiers. This method is asynchronous and needs a call to `getUpdateSimStatus` method in order to get result.
  - `getUpdateSimStatus`: Retrieve result of a `submitUpdateSimStatus`, according to a ticket number.
  - `submitConsumptionTrackingRequest`: Request for a set of SIM cards's traffic in a date range. This method is asynchronous and needs a call to `getConsumptionTracking` method in order to get result.
  - `getConsumptionTracking`: Retrieve result of a `submitConsumptionTrackingRequest` according to a ticket number.
  - `getNetworkStatus`: Get the network covering status into an area defined by its latitude and longitude.
  - `getIncidentDiagnostics`: Get the diagnostic analysis of a SIM card, including the network covering status of the SIM card.

- `submitSessionHistoryRequest`: Request for a SIM card's set of statistics by sessions in a date range. This method is asynchronous and needs a call to `getSessionHistory` method in order to get result.
- `getSessionHistory`: Retrieve result of a `submitSessionHistoryRequest` according to a ticket number.

**Figure 4.8. Malima UML class diagram.**



- `MalimaEndPointType`: this class represents an enumeration of all kinds of end points available for M2M service.

**Figure 4.9. MalimaEndPointType UML class diagram.**

### 3. Generated

The `com.orange.api.malima.generated` namespace contains classes automatically generated from the WSDL file by the `wsdl2java.class` tool included in the Axis Framework.

### 4. SDK configuration

Before requesting the M2M API through the SDK, you must initialize `Malima` and `MalimaUser` classes:

- settings for `Malima` class:
  - the web proxy: you can disable the web proxy, use the default Windows proxy or specify your own proxy.
  - the service URL.
- settings for `MalimaUser` class:
  - the access key and the access key password for HTTP basic auth.

You can do it either by using the `malima.properties` configuration file described above or by setting programmatically properties of the objects.

If you choose the configuration file way, you have to use the `loadFromConfigFile()` method (or one of its overrides):

- `loadFromConfigFile()`: it loads settings from the configuration file named `malima.properties` and located in the same folder than the current working executable.
- `loadFromConfigFile(String fileName)`: it loads settings from the configuration file whose path is `fileName`.
- `loadFromConfigFile(java.util.File fileStream)`: it loads settings from the stream pointed by `fileStream`.

An exception is thrown if the configuration file is not found or contains invalid data.

Here is an example of the configuration file way:

```
//Initializes the Malima service
```

```

Malima.loadFromConfigFile();

//Initializes the Malima User
MalimaUser malimaUser = new MalimaUser();
malimaUser.loadFromConfigFile();

```

If you choose the programmatical way, you have to set manually all the required object properties:

loadFromConfigFile initializeServiceURL

- properties for Malima class:
  - initializeProxy(String host, String port, String username, String password).
  - initializeService(String connectivityDirectoryUrl, String consumptionTrackingUrl, String incidentDiagnosticsUrl, String networkStatusUrl, String sessionHistoryUrl, String simLifecycleManagementUrl, String subscriptionStatusUrl).
  - initializeServerCertificate(String trustStore, String trustStorePassword).
- properties for MalimaUser class (from OrangeApiUser):
  - loadFromConfigFile(String path)
  - getClient(String endPointAddress, MalimaEndPointType endPointType)
  - setAccessKey(String accessKey).
  - setAccessKeyPassword(String accessKeyPassword).

Here is an example of the programmatical way:

```

String yourProxyHost = "yourProxyHost";
String yourProxyPort = "yourProxyPort";

// initialize Malima class
Malima.initializeProxy(yourProxyHost, yourProxyPort);

// (optional)
Malima.initializeService(connectivityDirectoryUrl,
    consumptionTrackingUrl, incidentDiagnosticsUrl, networkStatusUrl,
    sessionHistoryUrl, simLifecycleManagementUrl, subscriptionStatusUrl);

// (required)
Malima.initializeServerCertificate(
    "path_to_your_keystore.jks", "yourKeyStorePassowrd");

MalimaUser malimaUser = new MalimaUser();
malimaUser.setAccessKey("yourAccessKey");
malimaUser.setAccessKeyPassword("yourAccessKeyPassword");

```

## 5. Logging

The SDK uses the [log4j library](#) for purposes of application debugging and auditing. If you don't want to use this feature, just skip this section: it won't impact the execution of your program.

log4j is part of the [Apache Logging Services](#) project. log4j is a tool to help the programmer output log statements to a variety of output targets.

Typically the log4j configuration is specified using a properties file named `log4j.properties`.

For debugging purposes, please create a `log4j.properties` file in your classpath with the following content.

```
log4j.debug=true
log4j.rootLogger=debug,out1
log4j.logger.com.orange=debug,out2
log4j.logger.org.apache.axis.transport.http.HTTPSender=debug,inoutHTTP

# --- The 5 levels of threshold used by log4j ---
# debug
# info
# warn
# error
# fatal

# out1 is a console appender for System.out with an 'info' threshold.
log4j.appender.out1=org.apache.log4j.ConsoleAppender
log4j.appender.out1.Target=System.out
log4j.appender.out1.Threshold=info
log4j.appender.out1.layout=org.apache.log4j.SimpleLayout

# out2 is an Html file with an 'info' threshold.
log4j.appender.out2=org.apache.log4j.FileAppender
log4j.appender.out2.File=logs/out2.html
log4j.appender.out2.layout=org.apache.log4j.HTMLLayout
log4j.appender.out2.Threshold=info
log4j.appender.out2.Append=false
log4j.appender.out2.layout.LocationInfo=true
log4j.appender.out2.layout.Title=Logs for com

# inoutHTTP.log is a file with a maximal size of 1Mo.
# inoutHTTP.log has 2 saved file ( inoutHTTP.log.1 and inoutHTTP.log.2)
log4j.appender.inoutHTTP=org.apache.log4j.RollingFileAppender
log4j.appender.inoutHTTP.File=logs/inoutHTTP.log
log4j.appender.inoutHTTP.Append=true
log4j.appender.inoutHTTP.MaxFileSize=1MB
log4j.appender.inoutHTTP.MaxBackupIndex=2
log4j.appender.inoutHTTP.layout=org.apache.log4j.PatternLayout
log4j.appender.inoutHTTP.layout.ConversionPattern=%m%n
```

The output file `out2.html` and `inoutHTTP.log` will be stored in the logs folder created at the root level of your classpath.

Here is a sample of the content of the `inoutHTTP.log` file:

```
Enter: HTTPSender::invoke
XML sent:
-----
POST /mockISubscriptionStatus HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:8088
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 586
Authorization: Basic eW91c19sb2dpbjp5b3VyX3Bhc3N3b3Jk
```

```

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><getSubscriptionStatus
xmlns="http://webservice.malima.francetelecom.com/v1"><ns1:lineIdentifier xmlns:ns1="http://
subscriptionStatus.types.malima.francetelecom.com"><ns2:subscriptionNumber xmlns:ns2="http://
common.types.malima.francetelecom.com">23697128</ns2:subscriptionNumber></ns1:lineIdentifier></
getSubscriptionStatus></soapenv:Body></soapenv:Envelope>
HTTP/1.1 200 OK
Content-Type text/xml; charset=utf-8
Server Jetty(6.1.x)

no Content-Length

XML received:
-----
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v1="http://webservice.malima.francetelecom.com/v1" xmlns:sub="http://
subscriptionStatus.types.malima.francetelecom.com" xmlns:com="http://
common.types.malima.francetelecom.com"><soapenv:Body><v1:getSubscriptionStatusResponse>
<sub:customerEnvironmentIdentifier>12</sub:customerEnvironmentIdentifier>
<sub:sim>
<com:serialNumber>12</com:serialNumber>
</sub:sim>
<sub:subscription>
<com:identifier>12</com:identifier>
</sub:subscription>
</v1:getSubscriptionStatusResponse></soapenv:Body></soapenv:Envelope>
Exit: HTTPDispatchHandler::invoke
Enter: HTTPSender::invoke
XML sent:
-----
POST /mockISubscriptionStatus HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:8088
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 586
Authorization: Basic eW91c19sb2dpbjp5b3VyX3Bhc3N3b3Jk

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><getSubscriptionStatus
xmlns="http://webservice.malima.francetelecom.com/v1"><ns1:lineIdentifier xmlns:ns1="http://
subscriptionStatus.types.malima.francetelecom.com"><ns2:subscriptionNumber xmlns:ns2="http://
common.types.malima.francetelecom.com">23697128</ns2:subscriptionNumber></ns1:lineIdentifier></
getSubscriptionStatus></soapenv:Body></soapenv:Envelope>
HTTP/1.1 200 OK
Content-Type text/xml; charset=utf-8
Server Jetty(6.1.x)

no Content-Length

XML received:
-----
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v1="http://webservice.malima.francetelecom.com/v1" xmlns:sub="http://

```

```

subscriptionStatus.types.malima.francetelecom.com" xmlns:com="http://
common.types.malima.francetelecom.com"><soapenv:Body><v1:getSubscriptionStatusResponse>
  <sub:customerEnvironmentIdentifier>12</sub:customerEnvironmentIdentifier>
  <sub:sim>
  <com:serialNumber>12</com:serialNumber>
  </sub:sim>
  <sub:subscription>
  <com:identifier>12</com:identifier>
  </sub:subscription>
</v1:getSubscriptionStatusResponse></soapenv:Body></soapenv:Envelope>
Exit: HTTPDispatchHandler::invoke
Enter: HTTPSender::invoke
XML sent:
-----
POST /mockISubscriptionStatus HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: localhost:8088
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 586
Authorization: Basic eW91c19sb2dpbjp5b3VyX3Bhc3N3b3Jk

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><getSubscriptionStatus
xmlns="http://webservice.malima.francetelecom.com/v1"><ns1:lineIdentifier xmlns:ns1="http://
subscriptionStatus.types.malima.francetelecom.com"><ns2:subscriptionNumber xmlns:ns2="http://
common.types.malima.francetelecom.com">23697128</ns2:subscriptionNumber></ns1:lineIdentifier></
getSubscriptionStatus></soapenv:Body></soapenv:Envelope>
HTTP/1.1 200 OK
Content-Type text/xml; charset=utf-8
Server Jetty(6.1.x)

no Content-Length

XML received:
-----
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v1="http://webservice.malima.francetelecom.com/v1" xmlns:sub="http://
subscriptionStatus.types.malima.francetelecom.com" xmlns:com="http://
common.types.malima.francetelecom.com"><soapenv:Body><v1:getSubscriptionStatusResponse>
  <sub:customerEnvironmentIdentifier>12</sub:customerEnvironmentIdentifier>
  <sub:sim>
  <com:serialNumber>12</com:serialNumber>
  </sub:sim>
  <sub:subscription>
  <com:identifier>12</com:identifier>
  </sub:subscription>
</v1:getSubscriptionStatusResponse></soapenv:Body></soapenv:Envelope>
Exit: HTTPDispatchHandler::invoke

```

This is an example of the out2.html content file:

**Figure 4.10. Example: out2.html log file content.**

Time	Thread	Level	Category	File:Line	Message
0	main	INFO	com.orange.api.malima.test.local.Local	Local.java:45	Loading configuration at target/test-classes/malimaLocal.properties
94	main	DEBUG	com.orange.api.malima.MalimaUser	MalimaUser.java:132	MalimaUser.getClient for SubscriptionStatus on http://localhost:8088/mock/SubscriptionStatus

Log session start time Fri Feb 26 10:48:42 CET 2010

Time	Thread	Level	Category	File:Line	Message
0	main	INFO	com.orange.api.malima.test.local.Local	Local.java:45	Loading configuration at target/test-classes/malimaLocal.properties
94	main	DEBUG	com.orange.api.malima.MalimaUser	MalimaUser.java:132	MalimaUser.getClient for SubscriptionStatus on http://localhost:8088/mock/SubscriptionStatus

Log session start time Fri Feb 26 10:49:20 CET 2010

Time	Thread	Level	Category	File:Line	Message
0	main	INFO	com.orange.api.malima.test.local.Local	Local.java:45	Loading configuration at target/test-classes/malimaLocal.properties
125	main	DEBUG	com.orange.api.malima.MalimaUser	MalimaUser.java:132	MalimaUser.getClient for SubscriptionStatus on http://localhost:8088/mock/SubscriptionStatus

For more information, you can refer to the [Apache log4j manual](#).

## 6. First project

A "Hello World"-like WebService has been created to both test and check your machine environment and your connectivity to the M2M API. Please do the following steps:

- Generate the keystore with the keytool software.
- Create a new Java application project in your favorite IDE (Eclipse, Netbeans...).
- Add the malima-java-<version>.jar library to the build path of your project.
- Add the jars files contained in the zip file named as dependencies\_of\_malima-java\_<version>.zip to the build path of your project.
- Copy the malima.properties file at the root of the CLASSPATH of your project.
- Finally create a main java class. You can choose between the two followings with or without using the config file.
- Run your project, the response for the code snippets should give your connectivity directory for your chosen subscription number. Please refer to the one of the following code snippets to build your main class:

### 6.1. Example of a main class to test connectivity with hard coded configuration.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetSubscriptionStatusResponse;
import com.orange.api.malima.generated.types.LineIdentifier;

public class FirstAccessToMalima {

    // your credentials in order to access Malima services
    private static String accessKey = "your_access_key";
    private static String accessKeyPassword = "your_access_key_password";
    // one of your valid Malima Subscription Number
    private static String yourSubscriptionNumber = "your_subscription_number";

```

```

// the valid certificate path and password
private static String certificatePath = "your/path/to/certificate/orangeapi.jks";
private static String certificatePassword = "orangeapi";

public static void main(String[] args) {
    try {
        String yourProxyHost = null;
        String yourProxyPort = null;

        String connectivityDirectoryUrl = "your_connectivity_directory_url";
        String consumptiontrackingUrl = "your_consumption_tracking_url";
        String incidentDiagnosticsUrl = "your_incident_diagnostics_url";
        String networkStatusUrl = "your_network_status_url";
        String sessionHistoryUrl = "your_session_history_url";
        String simLifecycleManagementUrl = "your_sim_lifecycle_management_url";
        String subscriptionStatusUrl = "your_subscription_status_url";

        // (optional)
        Malima.initializeProxy(yourProxyHost, yourProxyPort);

        // (optional)
        Malima.initializeService(connectivityDirectoryUrl,
            consumptiontrackingUrl, incidentDiagnosticsUrl, networkStatusUrl,
            sessionHistoryUrl, simLifecycleManagementUrl, subscriptionStatusUrl);

        // (required)
        Malima.initializeServerCertificate(certificatePath, certificatePassword);

        MalimaUser malimaUser = new MalimaUser();
        malimaUser.setAccessKey(accessKey);
        malimaUser.setAccessKeyPassword(accessKeyPassword);
        // create the LineIdentifier you want to see
        LineIdentifier lineIdentifier = new LineIdentifier();
        // set the subscription number of the LineIdentifier
        lineIdentifier.setSubscriptionNumber(yourSubscriptionNumber);
        // Then call the service with requested LineIdentifier
        GetSubscriptionStatusResponse response = null;
        response = Malima.getSubscriptionStatus(malimaUser, lineIdentifier);

        // transform as a String
        StringBuffer result = new StringBuffer();
        if (response != null) {
            result.append("Customer Environnement : "
                + response.getCustomerEnvironmentIdentifier() + "\n");
            result.append("---\n");
            if (response.getSubscription() != null) {
                result.append("Subscription Id      : "
                    + response.getSubscription().getIdentifier() + "\n");
                result.append("Subscription creat.date: "
                    + response.getSubscription().getCreationDate().getTime() + "\n");
                result.append("---\n");
            }
            if (response.getSim() != null) {
                result.append("Sim SN              : "
                    + response.getSim().getSerialNumber() + "\n");
                result.append("Sim IMEI           : "
                    + response.getSim().getImei() + "\n");
                result.append("Sim status         : "
                    + response.getSim().getStatus() + "\n");
            }
        }
    }
}

```

```

        result.append("Sim request status      : "
            + response.getSim().getRequestedStatus() + "\n");
        result.append("----\n");
    }
    if (response.getDevice() != null) {
        result.append("Device SN                : "
            + response.getDevice().getSerialNumber() + "\n");
        result.append("Device identifier        : "
            + response.getDevice().getUniqueIdentifier() + "\n");
        result.append("Device description       : "
            + response.getDevice().getDescription() + "\n");
        result.append("Device category          : "
            + response.getDevice().getCategory() + "\n");
        result.append("Device address           : "
            + response.getDevice().getAddress() + "\n");
        result.append("----\n");
    }
    if (response.getMachine() != null) {
        result.append("Machine SN                : "
            + response.getMachine().getSerialNumber() + "\n");
        result.append("Machine name              : "
            + response.getMachine().getName() + "\n");
        result.append("Machine description       : "
            + response.getMachine().getDescription() + "\n");
        result.append("Machine update date      : "
            + response.getMachine().getUpdateDate().getTime() + "\n");
        result.append("----\n");
    }
    } else {
        result.append("No response returned\n");
    }
    }

    System.out
        .println("GetSubscriptionStatus response for Subscription Number "
            + yourSubscriptionNumber + " : \n" + result.toString());

    } catch (ConfigurationException configurationException) {
        configurationException.printStackTrace();
    } catch (FunctionalException functionalException) {
        functionalException.printStackTrace();
    } catch (TechnicalException technicalException) {
        technicalException.printStackTrace();
    }
}
}

```

## 6.2. Example of a main class to test connectivity with configuration file.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetSubscriptionStatusResponse;

```

```

import com.orange.api.malima.generated.types.LineIdentifier;

public class FirstAccessToMalimaWithConfigFile {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);
            // create the LineIdentifier you want to see
            LineIdentifier lineIdentifier = new LineIdentifier();
            // set the subscription number of the LineIdentifier
            lineIdentifier.setSubscriptionNumber(yourSubscriptionNumber);
            // Then call the service with requested LineIdentifier
            GetSubscriptionStatusResponse response = null;
            response = Malima.getSubscriptionStatus(malimaUser, lineIdentifier);

            // transform as a String
            StringBuffer result = new StringBuffer();
            if (response != null) {
                result.append("Customer Environnement : "
                    + response.getCustomerEnvironmentIdentifier() + "\n");
                result.append("---\n");
                if (response.getSubscription() != null) {
                    result.append("Subscription Id      : "
                        + response.getSubscription().getIdentifier() + "\n");
                    result.append("Subscription creat.date: "
                        + response.getSubscription().getCreationDate().getTime() + "\n");
                    result.append("---\n");
                }
                if (response.getSim() != null) {
                    result.append("Sim SN                : "
                        + response.getSim().getSerialNumber() + "\n");
                    result.append("Sim IMEI              : "
                        + response.getSim().getImei() + "\n");
                    result.append("Sim status            : "
                        + response.getSim().getStatus() + "\n");
                    result.append("Sim request status    : "
                        + response.getSim().getRequestedStatus() + "\n");
                    result.append("---\n");
                }
                if (response.getDevice() != null) {
                    result.append("Device SN             : "
                        + response.getDevice().getSerialNumber() + "\n");
                    result.append("Device identifier     : "
                        + response.getDevice().getUniqueIdentifier() + "\n");
                    result.append("Device description    : "
                        + response.getDevice().getDescription() + "\n");
                    result.append("Device category       : "
                        + response.getDevice().getCategory() + "\n");
                    result.append("Device address        : "
                        + response.getDevice().getAddress() + "\n");
                    result.append("---\n");
                }
                if (response.getMachine() != null) {
                    result.append("Machine SN           : "

```

```

        + response.getMachine().getSerialNumber() + "\n");
    result.append("Machine name      : "
        + response.getMachine().getName() + "\n");
    result.append("Machine description  : "
        + response.getMachine().getDescription() + "\n");
    result.append("Machine update date   : "
        + response.getMachine().getUpdateDate().getTime() + "\n");
    result.append("----\n");
    }
} else {
    result.append("No response returned\n");
}
System.out.println("SubscriptionStatus response for Subscription Number "
    + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The outputs look like:

```

GetSubscriptionStatus response for Subscription Number 23697256 :
Customer Environnement : 64095147
---
Subscription Id      : 23697256
Subscription creat.date: Wed Mar 17 00:00:00 CET 2010
---
Sim SN              : 1956473333658
Sim IMEI            : null
Sim status          : PRE_ACTIVATED
Sim request status  : null
---

```

Congratulations!

# Chapter 5. Methods

You should first make sure you've read the prerequisites before coding and running the following program listing. The prerequisites explain, among others, how to set up your proxy information and your account information through a configuration file.

## 1. Basic scenarios

In this chapter you will learn how to use the base objects we defined for you inside our SDK through some ready-to-use samples.

What are the basic steps to get your subscription status for a set of SIM cards ?

1. Create a `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifierCollection` for a set of SIM cards, based on their `SubscriptionIdentifier` for instance.
3. In the end, call the `GetSubscriptionStatus` API method. You will know everything about a set of SIM cards and their connectivity directory : their status, in which devices the SIM cards are set, in which machines the devices are set.

What are the basic steps to search in your connectivity directory ?

1. Create a `MalimaUser` and initialize its parameters.
2. If you don't have information about your SIM cards or serial numbers, you could Search in your Connectivity Directory according to a big range of searching criterions such as the state of searched SIM cards, the properties of a device of machine, dates of SIM cards changing states, phone numbers, ....
3. In the end, call the `SearchInConnectivityDirectory` API method. You will get the set of SIM cards corresponding to your searchin criterions.

What are the basic steps to get then update your connectivity directory ?

1. Create a `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifierCollection` for a set of SIM cards.
3. Call the `GetConnectivityDirectory` method.
4. Modify the `ConnectivityDirectory` array.
5. In the end, call the `UpdateConnectivityDirectory` API method using the modified `ConnectivityDirectory`.

What are the basic steps to update a SIM card status ?

1. Create an `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifierCollection` for a set of SIM cards.
3. Create a (`SimLifecycleManagement`) `SimStatus` corresponding to the requested updated status.
4. Create a `TestMode` for the area covered (`INTERNATIONAL`, `EUROPE`, `METROPOLE`, `AU_COMPTEUR`).
5. Call the asynchronous `SubmitUpdateSimStatus` API method.
6. Get the `TicketNumber` returned.
7. Call the `GetUpdateSimStatus` API method until the global response status of the method is either `TERMINATED` or `ERROR`, meaning the asynchronous modification has ended.

What are the basic steps to get consumption tracking of a set of SIM cards ?

1. Create an `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifierCollection` for a set of SIM cards.
3. Create a `start date` and an `end date` for a given period.
4. Call the asynchronous `SubmitConsumptionTracking` API method.
5. Get the `TicketNumber` returned.
6. Call the `GetConsumptionTracking` API method until the global response status of the method is either `TERMINATED` or `ERROR`, meaning the asynchronous generation has ended.

What are the basic steps to get your network status ?

1. Create a `MalimaUser` and initialize its parameters.
2. Create a `Coordinates` object corresponding to GPS coordinates of the area you where want the network status.
3. In the end, call the `GetNetworkStatus` API method. You will know everything about the network in the area, for each kind of network technology.

What are the basic steps to get incident diagnostics of a SIM card ?

1. Create a `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifier` for the SIM card.
3. In the end, call the `GetIncidentDiagnostics` API method. You will know everything about the state of the SIM card, the network status in the area, troubles occurring in the area.

What are the basic steps to get session history of a given SIM cards ?

1. Create an `MalimaUser` and initialize its parameters.
2. Create a `LineIdentifier` for the SIM card.
3. Create a `start date` and an `end date` for a given period.
4. Call the asynchronous `SubmitSessionHistory` API method.
5. Get the `TicketNumber` returned.
6. Call the `GetSessionHistory` API method until the global response status of the method is either `TERMINATED` or `ERROR`, meaning the asynchronous generation has ended.

## 2. GetSubscriptionStatus method

### 2.1. Description

Refresh SIM card's data and return the full data set information concerning one SIM card

### 2.2. Input parameter

**Table 5.1. getSubscriptionStatus: Input parameters.**

Name	Type	Description	Cardinality
<code>malimaUser</code>	<code>MalimaUser</code>	A Configurator for the Malima service	1..1
<code>lineIdentifier</code>	<code>LineIdentifier</code>	The SIM card identifier referenced by a <code>subscriptionNumber</code> , <code>simSerialNumber</code> , <code>deviceImei</code> , <code>deviceSerialNumber</code> , <code>deviceUniqueId</code> , <code>machineSerialNumber</code> , or a phone number associated to voice, data or fax.	1..1

## 2.3. Output parameter

**Table 5.2. getSubscriptionStatus: output parameters.**

Name	Type	Description	Cardinality
response	GetSubscriptionStatusResponse	A description of the SIM card data, made of a customer environment Id, a SIM card description, subscription state, device handling the SIM card description, machine handling the device description, an array of customer identifiers	1..1

## 2.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 2.5. Sample code

This sample shows how to Get Subscription Status of a SIM card.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetSubscriptionStatusResponse;
import com.orange.api.malima.generated.types.LineIdentifier;

public class SdkGetSubscriptionStatus {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            // create the LineIdentifier you want to see
            LineIdentifier lineIdentifier = new LineIdentifier();
            // set the subscription number of the LineIdentifier
            lineIdentifier.setSubscriptionNumber(yourSubscriptionNumber);
            // Then call the service with requested LineIdentifier
            GetSubscriptionStatusResponse response = null;
            response = Malima.getSubscriptionStatus(malimaUser, lineIdentifier);

            // transform as a String
            StringBuffer result = new StringBuffer();
            if (response != null) {
                result.append("Customer Environnement : "
                    + response.getCustomerEnvironmentIdentifier() + "\n");
            }
        } catch (TechnicalException | FunctionalException | ConfigurationException e) {
            e.printStackTrace();
        }
    }
}

```

```

result.append("---\n");
if (response.getSubscription() != null) {
    result.append("Subscription Id      : "
        + response.getSubscription().getIdentifier() + "\n");
    result.append("Subscription creat.date: "
        + response.getSubscription().getCreationDate().getTime() + "\n");
    result.append("---\n");
}
if (response.getSim() != null) {
    result.append("Sim SN                : "
        + response.getSim().getSerialNumber() + "\n");
    result.append("Sim IMEI             : "
        + response.getSim().getImei() + "\n");
    result.append("Sim status          : "
        + response.getSim().getStatus() + "\n");
    result.append("Sim request status  : "
        + response.getSim().getRequestedStatus() + "\n");
    result.append("---\n");
}
if (response.getDevice() != null) {
    result.append("Device SN           : "
        + response.getDevice().getSerialNumber() + "\n");
    result.append("Device identifier   : "
        + response.getDevice().getUniqueIdentifier() + "\n");
    result.append("Device description  : "
        + response.getDevice().getDescription() + "\n");
    result.append("Device category     : "
        + response.getDevice().getCategory() + "\n");
    result.append("Device address      : "
        + response.getDevice().getAddress() + "\n");
    result.append("---\n");
}
if (response.getMachine() != null) {
    result.append("Machine SN         : "
        + response.getMachine().getSerialNumber() + "\n");
    result.append("Machine name       : "
        + response.getMachine().getName() + "\n");
    result.append("Machine description : "
        + response.getMachine().getDescription() + "\n");
    result.append("Machine update date : "
        + response.getMachine().getUpdateDate().getTime() + "\n");
    result.append("---\n");
}
} else {
    result.append("No response returned\n");
}
System.out
    .println("GetSubscriptionStatus response for Subscription Number "
        + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}

```

The output looks like:

```

GetSubscriptionStatus response for Subscription Number 23697255 :
Customer Environnement : 64095147
---
Subscription Id      : 23697255
Subscription creat.date: Wed Mar 17 00:00:00 CET 2010
---
Sim SN              : 1956473384156
Sim IMEI            : null
Sim status          : PRE_ACTIVATED
Sim request status  : null
---

```

## 3. GetConnectivityDirectory method

### 3.1. Description

Get your Connectivity Directory information concerning a set of SIM cards.

### 3.2. Input parameter

**Table 5.3. getConnectivityDirectory: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
lineIdentifier Collection	LineIdentifier Collection	A collection of Line Identifiers	1..1

### 3.3. Output parameter

**Table 5.4. getConnectivityDirectory: output parameters.**

Name	Type	Description	Cardinality
response	Connectivity DirectoryResponse	An array of ConnectivityDirectory and a Collection of unknown LineIdentifiers	1..1

### 3.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

### 3.5. Sample code

This sample shows you how get a connectivity directory referential from a collection of line identifiers.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;

```

```

import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetConnectivityDirectoryResponse;
import com.orange.api.malima.generated.service.connectivitydirectory.type.ConnectivityDirectory;
import com.orange.api.malima.generated.types.LineIdentifierCollection;

public class SdkGetConnectivityDirectory {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            // create the LineIdentifier collection you want to see
            LineIdentifierCollection lineIdentifierCollection = new LineIdentifierCollection();
            // create the array where to set the subscription numbers
            String[] subscriptionNumbers = new String[1];
            // set the subscription number of the LineIdentifier
            subscriptionNumbers[0] = yourSubscriptionNumber;
            // refer the LineIdentifiers by their subscription number
            lineIdentifierCollection.setSubscriptionNumber(subscriptionNumbers);

            // Then call the service with requested LineIdentifier collection
            GetConnectivityDirectoryResponse response = null;
            response = Malima.getConnectivityDirectory(malimaUser, lineIdentifierCollection);

            StringBuffer result = new StringBuffer();
            if (response != null) {
                if (response.getConnectivityDirectory() != null) {
                    result.append("Number of Connectivity Directory found: "
                        + response.getConnectivityDirectory().length + "\n");
                    for (ConnectivityDirectory connectivityDirectory : response
                        .getConnectivityDirectory()) {
                        result.append(" ---\n");
                        result.append("    Customer Environment Id: "
                            + connectivityDirectory.getCustomerEnvironmentIdentifier()
                            + "\n");
                        result.append(" ---\n");
                        if (connectivityDirectory.getSim() != null) {
                            result.append("    Sim SN           : "
                                + connectivityDirectory.getSim().getSerialNumber() + "\n");
                            result.append("    Sim IMEI        : "
                                + connectivityDirectory.getSim().getImei() + "\n");
                            result.append("    Sim status      : "
                                + connectivityDirectory.getSim().getStatus() + "\n");
                            result.append("    Sim request status : "
                                + connectivityDirectory.getSim().getRequestedStatus() + "\n");
                            result.append("    ---\n");
                        }
                        if (connectivityDirectory.getDevice() != null) {
                            result.append("    Device SN       : "
                                + connectivityDirectory.getDevice().getSerialNumber() + "\n");
                            result.append("    Device Identifier : "
                                + connectivityDirectory.getDevice().getUniqueIdentifier()

```

```

        + "\n");
    result.append("    Device description : "
        + connectivityDirectory.getDevice().getDescription() + "\n");
    result.append("    Device category : "
        + connectivityDirectory.getDevice().getCategory() + "\n");
    result.append("    Device address : "
        + connectivityDirectory.getDevice().getAddress() + "\n");
    result.append("    ---\n");
}
if (connectivityDirectory.getMachine() != null) {
    result
        .append("    Machine SN : "
            + connectivityDirectory.getMachine().getSerialNumber()
            + "\n");
    result.append("    Machine name : "
        + connectivityDirectory.getMachine().getName() + "\n");
    result.append("    Machine description: "
        + connectivityDirectory.getMachine().getDescription() + "\n");
    result.append("    Machine update date: "
        + connectivityDirectory.getMachine().getUpdateDate().getTime() + "\n");
    result.append("    ---\n");
}
result.append("    ---\n");
}
result.append("----\n");
} else {
    result.append("No Connectivity Directory found\n");
}
} else {
    result.append("No response returned\n");
}

System.out
    .println("GetConnectivityDirectory response for Subscription Number "
        + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The output looks like:

```

GetConnectivityDirectory response for Subscription Number 23697248 :
Number of Connectivity Directory found: 1
---
Customer Environnement Id: 64095147
---
Sim SN           : 1956473384107
Sim IMEI         : null
Sim status       : PRE_ACTIVATED
Sim request status : null
---
Device SN       : DEVICE SN

```

```

Device Identifier : DEVICE_ID
Device description : DEVICE JAVA DESC
Device category : DEVICE JAVA CAT
Device address : DEVICE JAVA ADDR
---
Machine SN : MACHINE JAVA SN
Machine name : MACHINE JAVA DESC
Machine description: MACHINE JAVA DESC
Machine update date: Tue Mar 23 10:25:26 CET 2010
---
---
---
```

## 4. SearchInConnectivityDirectory method

### 4.1. Description

The SearchInConnectivityDirectory operation enable to search SIM cards according to search criterions.

### 4.2. Input parameter

**Table 5.5. searchInConnectivityDirectory: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
searchCriterion	Connectivity Directory SearchCriterion	The object containing the criterions. There can be up to two criterions	1..1
rangeSize	Integer	The max size returned by the method.	1..1
rangeStart	Integer	The index where the method must start from.	1..1

### 4.3. Output parameter

**Table 5.6. searchInConnectivityDirectory: output parameters.**

Name	Type	Description	Cardinality
response	SearchInConnectivity DirectoryResponse	The response made of an array of ConnectivityDirectory objects, the global number of results, and a boolean telling wether the number of results has exceeded or not	1..1

### 4.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

### 4.5. Sample code

This sample shows you how to Search in your Connectivity Directory.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.SearchInConnectivityDirectoryResponse;
import com.orange.api.malima.generated.service.connectivitydirectory.type.ConnectivityDirectory;
import
    com.orange.api.malima.generated.service.connectivitydirectory.type.ConnectivityDirectorySearchCriterion;
import com.orange.api.malima.generated.service.connectivitydirectory.type.SortAttribute;
import com.orange.api.malima.generated.service.connectivitydirectory.type.SortCriterion;
import com.orange.api.malima.generated.service.connectivitydirectory.type.SortOrder;
import com.orange.api.malima.generated.types.SimStatus;

public class SdkSearchInConnectivityDirectory {
    // private static String malimaSubscriptionNumberToSearchFor =
    // "your_subscription_number";
    private static String malimaSimSerialToSearchFor = "your_sim_serial_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            // create the searchCriterion : here the searchCriterion is a dummy
            // sample, but searchCriterion can be enhanced
            ConnectivityDirectorySearchCriterion searchCriterion = new
ConnectivityDirectorySearchCriterion();
            // searchCriterion.setSimSerialNumber(malimaSimSerialToSearchFor);
            searchCriterion.setSimStatus(new SimStatus[] { SimStatus.SLEEPING,
                SimStatus.ACTIVATED });
            // create the rangeSize and range start, response by blocks of 5
            // occurrences
            Integer rangeSize = new Integer(5);
            // starts offset at => 0
            Integer rangeStart = new Integer(0);
            // create sortCriteria : sort asc by status, then asc by SimSerial
            // Number
            SortCriterion firstSortCriterion = new SortCriterion();
            firstSortCriterion.setAttribute(SortAttribute.SIM_STATUS);
            firstSortCriterion.setOrder(SortOrder.ASC);
            SortCriterion secondSortCriterion = new SortCriterion();
            secondSortCriterion.setAttribute(SortAttribute.SIM_SSN);
            secondSortCriterion.setOrder(SortOrder.ASC);
            SortCriterion[] sortCriteria = new SortCriterion[] {
                firstSortCriterion, secondSortCriterion };

            // then call the search method
            SearchInConnectivityDirectoryResponse response = null;
            response = Malima.searchInConnectivityDirectory(malimaUser,
                searchCriterion, rangeSize, rangeStart, sortCriteria);

```

```

StringBuffer result = new StringBuffer();
if (response != null) {
    if (!response.isResultsNumberExceeded()) {
        result.append("Number of Connectivity Directory found: "
            + response.getTotalResultsNumber() + "\n");
        if (response.getConnectivityDirectory() != null) {
            for (ConnectivityDirectory connectivityDirectory : response
                .getConnectivityDirectory()) {
                result.append(" ---\n");
                result.append(" Customer Environment Id: "
                    + connectivityDirectory.getCustomerEnvironmentIdentifier()
                    + "\n");
                result.append(" ---\n");
                if (connectivityDirectory.getSubscription() != null) {
                    result.append(" Subscription Number: "
                        + connectivityDirectory.getSubscription().getIdentifier()
                        + "\n");
                    result.append(" ---\n");
                }

                if (connectivityDirectory.getSim() != null) {
                    result.append(" Sim SN : "
                        + connectivityDirectory.getSim().getSerialNumber() + "\n");
                    result.append(" Sim IMEI : "
                        + connectivityDirectory.getSim().getImei() + "\n");
                    result.append(" Sim status : "
                        + connectivityDirectory.getSim().getStatus() + "\n");
                    result.append(" Sim request status : "
                        + connectivityDirectory.getSim().getRequestedStatus()
                        + "\n");
                    result.append(" ---\n");
                }
            }
            if (connectivityDirectory.getDevice() != null) {
                result.append(" Device SN : "
                    + connectivityDirectory.getDevice().getSerialNumber()
                    + "\n");
                result.append(" Device Identifier : "
                    + connectivityDirectory.getDevice().getUniqueIdentifier()
                    + "\n");
                result
                    .append(" Device description : "
                        + connectivityDirectory.getDevice().getDescription()
                        + "\n");
                result.append(" Device category : "
                    + connectivityDirectory.getDevice().getCategory() + "\n");
                result.append(" Device address : "
                    + connectivityDirectory.getDevice().getAddress() + "\n");
                result.append(" ---\n");
            }
            if (connectivityDirectory.getMachine() != null) {
                result.append(" Machine SN : "
                    + connectivityDirectory.getMachine().getSerialNumber()
                    + "\n");
                result.append(" Machine name : "
                    + connectivityDirectory.getMachine().getName() + "\n");
                result.append(" Machine description: "
                    + connectivityDirectory.getMachine().getDescription()
                    + "\n");
                result.append(" Machine update date: "
                    + connectivityDirectory.getMachine().getUpdateDate()

```

```

        .getTime() + "\n");
        result.append("    ---\n");
    }
    result.append("    ---\n");
}
result.append("---\n");
} else {
    result.append("No Connectivity Directory found\n");
}
} else {
    result.append("Nb of returned Connectivity has exceeded\n");
}
} else {
    result.append("No response returned\n");
}
}
System.out
.println("SearchInConnectivityDirectory response for Search Criterion concerning Sim
Serial Number "
        + malimaSimSerialToSearchFor + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The output looks like:

```

SearchInConnectivityDirectory response for Search Criterion concerning SimSerial Number
1956473384107 :
Number of Connectivity Directory found: 2
---
Customer Environnement Id: 64095147
---
Subscription Number: 23697262
---
Sim SN           : 1956473333682
Sim IMEI         : null
Sim status       : ACTIVATED
Sim request status : null
---
---
Customer Environnement Id: 64095147
---
Subscription Number: 23697260
---
Sim SN           : 1956473333674
Sim IMEI         : null
Sim status       : SLEEPING
Sim request status : null
---
---
---

```

## 5. UpdateConnectivityDirectory method

### 5.1. Description

The UpdateConnectivityDirectory operation updates a set of connectivity directory data, managed by different kinds of identifiers.

### 5.2. Input parameter

**Table 5.7. updateConnectivityDirectory: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
connectivityDirectory	Update Connectivity Directory	The set of connectivity directory data, managed by different kinds of identifiers (SimSerialNumber, DeviceImei, DeviceSerialNumber, DeviceUniqueId, MachineSerialNumber, MsisdnVoice, MsisdnData and/or MsisdnFax).	1..1

### 5.3. Output parameter

**Table 5.8. updateConnectivityDirectory: output parameters.**

Name	Type	Description	Cardinality
response	UpdateConnectivityDirectoryResponse	An array of connectivity directories updated successfully, and an array of connectivity directories update failed	1..1

### 5.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

### 5.5. Sample code

This sample shows you how to Update a set of Connectivity Directory.

```
package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetConnectivityDirectoryResponse;
import com.orange.api.malima.generated.service.UpdateConnectivityDirectory;
import com.orange.api.malima.generated.service.UpdateConnectivityDirectoryResponse;
import com.orange.api.malima.generated.service.connectivitydirectory.type.ConnectivityDirectory;
```

```

import
  com.orange.api.malima.generated.service.connectivitydirectory.type.DataBySubscriptionNumber;
import com.orange.api.malima.generated.service.connectivitydirectory.type.UpdateFailed;
import com.orange.api.malima.generated.types.CommunicationModule;
import com.orange.api.malima.generated.types.Contact;
import com.orange.api.malima.generated.types.Device;
import com.orange.api.malima.generated.types.LineIdentifier;
import com.orange.api.malima.generated.types.LineIdentifierCollection;
import com.orange.api.malima.generated.types.Machine;

public class SdkUpdateConnectivityDirectory {
  private static String yourSubscriptionNumber = "your_subscription_number";

  public static void main(String[] args) {
    try {
      // Create and initialize a malimaUser for Malima
      // Service from specified configuration
      String pathToConfigFile = "your/path/to/configFile.properties";
      MalimaUser malimaUser = new MalimaUser();
      malimaUser.loadFromConfigFile(pathToConfigFile);
      Malima.loadFromConfigFile(pathToConfigFile);

      /**
       * call getConnectivityDirectory method in order to manage a
       * ConnectivityDirectory object
       */
      // create the LineIdentifier collection you want to see
      LineIdentifierCollection lineIdentifierCollection = new LineIdentifierCollection();
      // create the array where to set the subscription numbers
      String[] subscriptionNumbers = new String[1];
      // set the subscription number of the LineIdentifier
      subscriptionNumbers[0] = yourSubscriptionNumber;
      // refer the LineIdentifiers by their subscription number
      lineIdentifierCollection.setSubscriptionNumber(subscriptionNumbers);

      // Then call the service with requested LineIdentifier collection
      GetConnectivityDirectoryResponse getConnectivityDirectoryResponse = null;
      getConnectivityDirectoryResponse = Malima.getConnectivityDirectory(malimaUser,
        lineIdentifierCollection);

      StringBuffer result = new StringBuffer();

      // if there is a significant response, the UpdateConnectivityDirectory
      // method can be called
      if (getConnectivityDirectoryResponse.getConnectivityDirectory() != null
        && getConnectivityDirectoryResponse.getConnectivityDirectory().length != 0) {

        /** make changes in the ConnectivityDirectory */
        // create the UpdateConnectivityDirectory you want to update
        UpdateConnectivityDirectory updateConnectivityDirectory = new
UpdateConnectivityDirectory();
        // UpdateConnectivityDirectory will be managed by its
        // DataBySubscriptionNumber[]
        DataBySubscriptionNumber dataBySubscriptionNumber = new DataBySubscriptionNumber();
        // get back first connectivity directory subscription number, device and
        // machine from the getConnectivityDirectoryResponse
        String subscriptionNumber = getConnectivityDirectoryResponse
          .getConnectivityDirectory()[0].getSubscription().getIdentifier();
        Device device = getConnectivityDirectoryResponse
          .getConnectivityDirectory()[0].getDevice();

```

```

Machine machine = getConnectivityDirectoryResponse
    .getConnectivityDirectory()[0].getMachine();
// change data into device and machine (this is the goal of an update !)
if (device == null) {
    device = new Device();
}
device.setSerialNumber("My Java device SN");
device.setDescription("My Java device desc");
device.setUniqueIdentifier("My Java unique device id");
device.setCategory("My Java device cat");
device.setAddress("My Java device addr");
Contact contact = new Contact();
contact.setEmail("m2m@orange.com");
contact.setName("M2M");
contact.setPhone("0612345678");
device.setContact(contact);
CommunicationModule comMod = new CommunicationModule();
comMod.setModel("Java model");
comMod.setBrand("java TM");
device.setCommunicationModule(comMod);

if (machine == null) {
    machine = new Machine();
}
machine.setSerialNumber("My java machine SN");
machine.setDescription("My java machine desc");
machine.setName("My java machine name");

// assign subscriptionNumber, machine and device
dataBySubscriptionNumber.setSubscriptionNumber(subscriptionNumber);
// set updated device and machine
dataBySubscriptionNumber.setDeviceData(device);
dataBySubscriptionNumber.setMachineData(machine);
// fill updateConnectivityDirectory
DataBySubscriptionNumber[] databySubscriptionNumberArray = new DataBySubscriptionNumber[]
{ dataBySubscriptionNumber };
updateConnectivityDirectory
    .setDataBySubscriptionNumber(databySubscriptionNumberArray);

// Then call the service with requested UpdateConnectivityDirectory and
// get the response
UpdateConnectivityDirectoryResponse response = Malima
    .updateConnectivityDirectory(malimaUser,
        updateConnectivityDirectory);

if (response != null) {
    if (response.getUpdatesCompleted() != null
        && response.getUpdatesCompleted().getLineIdentifier() != null) {
        result.append("Number of completed Line Identifiers found: "
            + response.getUpdatesCompleted().getLineIdentifier().length
            + "\n");
        for (LineIdentifier lineIdentifier : response.getUpdatesCompleted()
            .getLineIdentifier()) {
            result.append(" Subscription Number      : "
                + lineIdentifier.getSubscriptionNumber() + "\n");
            result.append(" ---\n");
        }
    } else {
        result.append("No LineIdentifier completed\n");
    }
}

```

```

result.append("---\n");

if (response.getUpdatesFailed() != null
    && response.getUpdatesFailed().getFail() != null) {
    result.append("Number of UpdateFailed found: "
        + response.getUpdatesFailed().getFail().length + "\n");
    for (UpdateFailed updateFailed : response.getUpdatesFailed()
        .getFail()) {
        if (updateFailed.getLineIdentifier() != null) {
            result.append("  Subscription Number      : "
                + updateFailed.getLineIdentifier().getSubscriptionNumber()
                + "\n");
        } else {
            result.append("  Subscription Number      : unknown !!!\n");
        }
        result.append("  Error code                : "
            + updateFailed.getErrorCode() + "\n");
        result.append("  Error message             : "
            + updateFailed.getErrorMessage() + "\n");
        result.append("  ---\n");
    }
} else {
    result.append("No LineIdentifier failed\n");
}

// get the updated LineIdentifier collection you want to see
getConnectivityDirectoryResponse = Malima.getConnectivityDirectory(malimaUser,
lineIdentifierCollection);

if (response != null) {
    if (getConnectivityDirectoryResponse.getConnectivityDirectory() != null) {
        result.append("Number of Connectivity Directory found: "
            + getConnectivityDirectoryResponse.getConnectivityDirectory().length + "\n");
        for (ConnectivityDirectory connectivityDirectory : getConnectivityDirectoryResponse
            .getConnectivityDirectory()) {
            result.append("---\n");
            result.append("  Customer Environnement Id: "
                + connectivityDirectory.getCustomerEnvironmentIdentifier()
                + "\n");
            result.append("  ---\n");
            if (connectivityDirectory.getSim() != null) {
                result.append("    Sim SN                  : "
                    + connectivityDirectory.getSim().getSerialNumber() + "\n");
                result.append("    Sim IMEI                : "
                    + connectivityDirectory.getSim().getImei() + "\n");
                result.append("    Sim status              : "
                    + connectivityDirectory.getSim().getStatus() + "\n");
                result.append("    Sim request status     : "
                    + connectivityDirectory.getSim().getRequestedStatus() + "\n");
                result.append("    ---\n");
            }
            if (connectivityDirectory.getDevice() != null) {
                result.append("    Device SN              : "
                    + connectivityDirectory.getDevice().getSerialNumber() + "\n");
                result.append("    Device Identifier       : "
                    + connectivityDirectory.getDevice().getUniqueIdentifier()
                    + "\n");
                result.append("    Device description     : "
                    + connectivityDirectory.getDevice().getDescription() + "\n");
                result.append("    Device category        : "

```



```

Sim status      : PRE_ACTIVATED
Sim request status : null
---
Device SN       : My Java device SN
Device Identifier : My Java unique device id
Device description : My Java device desc
Device category  : My Java device cat
Device address   : My Java device addr
---
Machine SN      : My java machine SN
Machine name     : My java machine name
Machine description: My java machine desc
Machine update date: Tue Mar 23 10:32:12 CET 2010
---
---
---
```

## 6. SubmitUpdateSimStatus and GetUpdateSimStatus methods

### 6.1. SubmitUpdateSimStatus method

#### 6.1.1. Description

Submit an update of SIM cards's status for a set of SIM cards.

#### Tip

The method `SubmitUpdateSimStatus` is asynchronous and has to be used prior the `GetUpdateSimStatus` method call which gets back the submit result.

#### 6.1.2. Input parameter

**Table 5.9. submitUpdateSimStatus: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
lineIdentifier Collection	LineIdentifier Collection	A collection of Line Identifiers	1..1
SimSimStatus	requestedStatus	Can be ACTIVATED, ACTIVATED_FOR_TEST, SLEEPING or SUSPENDED	1..1
TestMode	testMode	Can be INTERNATIONAL, EUROPE, METROPOLE or AU_COMPTEUR. testMode must be set to null when requestedStatus is different from ACTIVATED_FOR_TEST	1..1

### 6.1.3. Output parameter

**Table 5.10. submitUpdateSimStatus: output parameters.**

Name	Type	Description	Cardinality
response	SubmitUpdateSim StatusRequest Response	A ticket number usable in the getUpdateSimStatus request, and a collection of unknown requested LineIdentifiers if any.	1..1

### 6.1.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 6.2. GetUpdateSimStatus method

### 6.2.1. Description

Get the result of a `SubmitUpdateSimStatus` method call.

#### Tip

The method `GetUpdateSimStatus` has to be used after a `SubmitUpdateSimStatus` method call, in order to get the asynchronous submit result.

As the method is the result of a preliminar asynchronous submit call, the result of the get method contains a `Ticket Status` telling wether the submit call work is finished, in progress or not started yet.

### 6.2.2. Input parameter

**Table 5.11. getUpdateSimStatus: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
ticketNumber	int	A ticket number returned by the <code>submitUpdateSimStatus</code> method call.	1..1

### 6.2.3. Output parameter

**Table 5.12. getUpdateSimStatus: output parameters.**

Name	Type	Description	Cardinality
response	GetUpdateSim StatusResultResponse	A global ticket status telling wether the request is being processed, in progress or terminated ; an array of <code>StatusUpdate</code> (one for each requested <code>LineIdentifier</code> ); a reminder of <code>SlmSimStatus</code> and <code>TestMode</code> values ; an array of unknown line identifier if any.	1..1

## 6.2.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 6.3. Sample code

This sample shows how to Submit then Get SIM Status Update of a SIM card.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetUpdateSimStatusResultResponse;
import com.orange.api.malima.generated.service.SubmitUpdateSimStatusRequestResponse;
import com.orange.api.malima.generated.service.simlifecyclemanagement.types.SlmSimStatus;
import com.orange.api.malima.generated.service.simlifecyclemanagement.types.StatusUpdate;
import com.orange.api.malima.generated.types.LineIdentifierCollection;
import com.orange.api.malima.generated.types.TestMode;
import com.orange.api.malima.generated.types.TicketStatusType;

public class SdkSubmitAndGetUpdateSimStatus {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            /** call SubmitUpdateSimStatus method **/
            // create the LineIdentifier collection you want to see
            LineIdentifierCollection lineIdentifierCollection = new LineIdentifierCollection();
            // create the array where to set the subscription numbers
            String[] subscriptionNumbers = new String[1];
            // set the subscription number of the LineIdentifier
            subscriptionNumbers[0] = yourSubscriptionNumber;
            // refer the LineIdentifiers by their subscription number
            lineIdentifierCollection.setSubscriptionNumber(subscriptionNumbers);
            // set the requested status to ACTIVATED FOR TEST ( This can be dangerous
            // or unfeasible according to the current state of your SIM !)
            SlmSimStatus requestedStatus = SlmSimStatus.ACTIVATED;
            // set the test mode to null, as the SimStatus is different of ACTIVATED_FOR_TEST
            TestMode testMode = null;

            StringBuffer result = new StringBuffer();
            result.append("==> SUBMIT METHOD CALL :\n");
            // Then call the service with requested LineIdentifier collection
            SubmitUpdateSimStatusRequestResponse submitResponse = null;
            submitResponse = Malima.submitUpdateSimStatus(malimaUser,

```

```

lineIdentifierCollection, requestedStatus, testMode);

// if there is a significant response, the GetUpdateSimStatus method will
// be called
if (submitResponse != null) {
    int ticketNumber = submitResponse.getTicketNumber();
    if (ticketNumber > -1) {
        result.append("Ticket number returned : " + ticketNumber);
        if (submitResponse.getUnknownLineIdentifiers() != null) {
            result.append("-----\n");
            result.append("Unknown Line Ids found : \n");
            if (submitResponse.getUnknownLineIdentifiers()
                .getSubscriptionNumber() != null) {
                for (String subsId : submitResponse.getUnknownLineIdentifiers()
                    .getSubscriptionNumber()) {
                    result.append(" " + subsId + "\n");
                }
            }
        }
    }
}
/*****/
/** call GetUpdateSimStatus method */
/*****/
GetUpdateSimStatusResultResponse getResponse = null;
TicketStatusType ticketStatus = TicketStatusType.WAITING;

result.append("\n");
result.append("=====> GET METHOD CALL :\n");
// get update sim status according to the ticket number return by the
// submit method
while ((!ticketStatus.equals(TicketStatusType.TERMINATED))
    && (!ticketStatus.equals(TicketStatusType.ERROR))) {
    // you could wait for a while here ...
    try{
        Thread.currentThread().sleep(30000); //sleep for 30 seconds
    }
    catch (InterruptedException ie){
        result.append("WARN : GetUpdateSimStatus interrupted while sleeping 30 secs" );
    }
    getResponse = Malima.getUpdateSimStatus(malimaUser, ticketNumber);
    if (getResponse != null) {
        ticketStatus = getResponse.getGlobalProcessingStatus();
    } else {
        ticketStatus = TicketStatusType.ERROR;
    }
} // while get not terminated

result.append("Ticket Status : " + ticketStatus + "\n");
// there is a response, check if the response is fine
if (getResponse != null
    && getResponse.getGlobalProcessingStatus().equals(
        TicketStatusType.TERMINATED)) {
    result.append("Process state returned : "
        + getResponse.getGlobalProcessingStatus().getValue() + "\n");
    for (StatusUpdate statusUpdate : getResponse.getStatusUpdate()) {
        if (statusUpdate.getInputLineIdentifier() != null) {
            result.append(" Line Id subs. Number : "
                + statusUpdate.getInputLineIdentifier()
                .getSubscriptionNumber() + "\n");
        } else {

```

```

        result.append("  Line Id subs. Number : unknown \n");
    }
    result.append("  Progress message      : "
        + statusUpdate.getProgressMessage() + "\n");
    result.append("  Error code - message : "
        + statusUpdate.getErrorCode() + " - "
        + statusUpdate.getErrorMessage() + "\n");
    if (statusUpdate.getUpdateDate() != null) {
        result.append("  Update date          : "
            + statusUpdate.getUpdateDate().getTime() + "\n");
    }
    result.append("  ----\n");
    }
    } else {
    result
        .append("No get update sim status response.\n");
    }
    } else {
    result.append("No ticket number returned\n");
    }
    } else {
    result.append("No response returned\n");
    }
    }

    System.out.println("UpdateSimStatus response for Subscription Number "
        + yourSubscriptionNumber + " : \n" + result.toString());

    } catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
    } catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
    } catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
    }
}
}

```

The output looks like:

```

UpdateSimStatus response for Subscription Number 23697260 :
==> SUBMIT METHOD CALL :
Ticket number returned : 541
=====> GET METHOD CALL :
Ticket Status : TERMINATED
Process state returned : TERMINATED
  Line Id subs. Number : 23697260
  Progress message      : null
  Error code - message : null - null
  Update date          : Tue Mar 23 10:54:47 CET 2010
  ----

```

## 7. SubmitConsumptionTracking and GetConsumptionTracking methods

### 7.1. SubmitConsumptionTracking method

#### 7.1.1. Description

Submit an Consumption Tracking request for a set of SIM cards in a given period of time. This method enables to consult a full traffic exchanged between a set of SIM cards, devices or machines and the Orange network.

#### Tip

The method `SubmitConsumptionTracking` is asynchronous and has to be used prior the `getConsumptionTracking` method call which gets back the submit result.

#### 7.1.2. Input parameter

Table 5.13. `submitConsumptionTracking`: Input parameters.

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
lineIdentifier Collection	LineIdentifier Collection	A collection of Line Identifiers	1..1
startDate	Calendar	The start date of the request	1..1
endDate	Calendar	The end date of the request	1..1

#### 7.1.3. Output parameter

Table 5.14. `submitConsumptionTracking`: output parameters.

Name	Type	Description	Cardinality
response	SubmitConsumption TrackingRequest Response	A ticket number usable in the <code>getConsumptionTracking</code> request, and a collection of unknown requested LineIdentifiers if any.	1..1

#### 7.1.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

### 7.2. GetConsumptionTracking method

#### 7.2.1. Description

Get the result of a `SubmitConsumptionTracking` method call.

## Tip

The method `GetConsumptionTracking` has to be used after a `SubmitConsumptionTracking` method call, in order to get the asynchronous submit result.

As the method is the result of a preliminar asynchronous submit call, the result of the get method contains a Ticket Status telling wether the submit call work is finished, in progress or not started yet.

### 7.2.2. Input parameter

**Table 5.15. `getConsumptionTracking`: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
ticketNumber	int	A ticket number returned by the <code>submitConsumptionTracking</code> method call.	1..1

### 7.2.3. Output parameter

**Table 5.16. `getConsumptionTracking`: output parameters.**

Name	Type	Description	Cardinality
response	<code>GetConsumptionTrackingResultResponse</code>	A global ticket status telling wether the request is being processed, in progress or terminated ; an array of Customer Identifier ; an array of Billing Account ; an array of Line (i.e. volume of consumption, i.e. CDRs for each Line) ; the global number of CDRs returned, the first CDR date and last CDR date, and an array of unknown line identifier if any.	1..1

### 7.2.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 7.3. Sample code

This sample shows how to Submit then Get Consumption Tracking of a set of SIM cards.

```

package com.orange.api.malima.doc.sdkGuide;

import java.util.Calendar;
import java.util.GregorianCalendar;
import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetConsumptionTrackingResponse;
import com.orange.api.malima.generated.service.SubmitConsumptionTrackingRequestResponse;
import com.orange.api.malima.generated.service.consumptiontracking.types.BillingAccount;
import com.orange.api.malima.generated.service.consumptiontracking.types.Line;

```

```

import com.orange.api.malima.generated.service.consumptiontracking.types.Volume;
import com.orange.api.malima.generated.types.LineIdentifierCollection;
import com.orange.api.malima.generated.types.TicketStatusType;

public class SdkSubmitAndGetConsumptionTracking {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            /*****
            /** call SubmitConsumptionTracking method **/
            /*****/
            // create the LineIdentifier collection you want to see
            LineIdentifierCollection lineIdentifierCollection = new LineIdentifierCollection();
            // create the array where to set the subscription numbers
            String[] subscriptionNumbers = new String[1];
            // set the subscription number of the LineIdentifier
            subscriptionNumbers[0] = yourSubscriptionNumber;
            // refer the LineIdentifiers by their subscription number
            lineIdentifierCollection.setSubscriptionNumber(subscriptionNumbers);
            // set the starting date and ending date of the session history
            Calendar startDateTime = new GregorianCalendar();
            Calendar endDateTime = new GregorianCalendar();
            startDateTime.set(2010, 0, 1); // jan, 1st 2010
            endDateTime.set(2010, 1, 1); // feb, 1st 2010

            StringBuffer result = new StringBuffer();
            result.append("==> SUBMIT METHOD CALL :\n");
            // Then call the service with requested LineIdentifier collection
            SubmitConsumptionTrackingRequestResponse submitResponse = null;
            submitResponse = Malima.submitConsumptionTracking(malimaUser,
                lineIdentifierCollection, startDateTime, endDateTime);

            // if there is a significant response, the GetUpdateSimStatus method can
            // be called
            if (submitResponse != null) {
                int ticketNumber = submitResponse.getTicketNumber();
                if (ticketNumber > -1) {
                    result.append("Ticket number returned : " + ticketNumber);
                    if (submitResponse.getUnknownLineIdentifiers() != null) {
                        result.append("-----\n");
                        result.append("Unknown Line Ids found : \n");
                        if (submitResponse.getUnknownLineIdentifiers()
                            .getSubscriptionNumber() != null) {
                            for (String subsId : submitResponse.getUnknownLineIdentifiers()
                                .getSubscriptionNumber()) {
                                result.append(" " + subsId + "\n");
                            }
                        }
                    }
                }
            }

            /*****
            /** call GetConsumptionTracking method **/

```

```

/*****/
result.append("\n");
result.append("=====> GET METHOD CALL :\n");

GetConsumptionTrackingResponse getResponse = null;
TicketStatusType ticketStatus = TicketStatusType.WAITING;

// get update sim status according to the ticket number return by the
// submit method
while ((!ticketStatus.equals(TicketStatusType.TERMINATED))
    && (!ticketStatus.equals(TicketStatusType.ERROR))) {
    // you could wait for a while here ...
    try{
        Thread.currentThread().sleep(5000); //sleep for 5 seconds
    }
    catch(InterruptedException ie){
        result.append("WARN : GetConsumptionTracking interrupted while sleeping 5 secs" );
    }
    getResponse = Malima.getConsumptionTracking(malimaUser,
        ticketNumber);
    if (getResponse != null) {
        ticketStatus = getResponse.getGlobalProcessingStatus();
    } else {
        ticketStatus = TicketStatusType.ERROR;
    }
} // while get not terminated

result.append("Ticket Status : " + ticketStatus + "\n");
// there is a response, check if the response is fine
if (getResponse != null
    && getResponse.getGlobalProcessingStatus() != null
    && getResponse.getGlobalProcessingStatus().getValue().equals(
        TicketStatusType.TERMINATED)) {
    result.append("Process value returned : "
        + getResponse.getGlobalProcessingStatus().getValue() + "\n");
    if (getResponse.getTotalCdrNumber() != null) {
        result.append("----\n");
        result.append("Total CDR number      : "
            + getResponse.getTotalCdrNumber() + "\n");
    }
    if (getResponse.getLine() != null) {
        result.append("----\n");
        result.append("Lines (nb)                : "
            + getResponse.getLine().length + "\n");
        for (Line line : getResponse.getLine()) {
            if (line.getInputLineIdentifier() != null) {
                result.append("  Line identifier        : "
                    + line.getInputLineIdentifier().getSubscriptionNumber()
                    + "\n");
            }
            if (line.getLineConsumptionTracking() != null) {
                result
                    .append("  Line consTrack cdrNb : "
                        + line.getLineConsumptionTracking().getCdrNumber()
                        + "\n");
            }
        }
    }
} else {
    result.append("----\n");
    result.append("No Lines found\n");
}

```

```

}
if (getResponse.getBillingAccount() != null
    && getResponse.getBillingAccount().length != 0) {
result.append("----\n");
result.append("Billing accounts      : \n");
for (BillingAccount billingAccount : getResponse
    .getBillingAccount()) {
result.append("  Account number      : "
    + billingAccount.getBillingAccountNumber() + "\n");
result.append("  Account Address      : "
    + billingAccount.getBillingAddress()
    .getStreetNameAndNumber() + " - "
    + billingAccount.getBillingAddress().getTown() + "\n");
if (billingAccount.getLinesConsumptionTrackingSynthesis() != null) {
result.append("    CDR number        : "
    + billingAccount.getLinesConsumptionTrackingSynthesis()
    .getCdrNumber() + "\n");
}
if (billingAccount.getLinesConsumptionTrackingSynthesis()
    .getVolume() != null) {
result.append("  Volume (nb)          : "
    + billingAccount.getLinesConsumptionTrackingSynthesis()
    .getVolume().length + "\n");
for (Volume volume : billingAccount
    .getLinesConsumptionTrackingSynthesis().getVolume()) {
result.append("    Value              : "
    + volume.getValue() + "\n");
if (volume.getType() != null) {
result.append("      Type              : "
    + volume.getType().getValue() + "\n");
}
if (volume.getBearer() != null) {
result.append("        Bearer          : "
    + volume.getBearer().getValue() + "\n");
}
if (volume.getRoaming() != null) {
result.append("          Roaming       : "
    + volume.getRoaming().getValue() + "\n");
}
result.append("    ----\n");
}
} else {
result.append("  ----\n");
result.append("  No Volume found      : \n");
}
}
if (billingAccount.getLinesReferences() != null) {
result.append("  ----\n");
result.append("  Line references      : \n");
if (billingAccount.getLinesReferences()
    .getSubscriptionNumber() != null) {
for (String subsId : billingAccount.getLinesReferences()
    .getSubscriptionNumber()) {
result
    .append("    Subscription number: " + subsId + "\n");
result.append("    ----\n");
}
}
} else {
result.append("  ----\n");
result.append("  No Line refs found   : \n");
}
}

```

```

        }
        result.append(" ----\n");
    }
    } else {
        result.append("----\n");
        result.append("No billing accounts found      : \n");
    }
    } else {
        result
            .append("No consumption tracking found.\n");
    }
    } else {
        result.append("No ticket number returned\n");
    }
    } else {
        result.append("No response returned\n");
    }
}

System.out
    .println("Consumption tracking response for Subscription Number "
        + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The output looks like:

```

SessionHistory response for Subscription Number 23697249 :
==> SUBMIT METHOD CALL :
Ticket number returned : 569
=====> GET METHOD CALL :
Ticket Status : TERMINATED
No consumption tracking found.

```

## 8. GetNetworkStatus method

### 8.1. Description

Get Network Status information from given GPS coordinates, i.e. in a given geographic area.

## 8.2. Input parameter

**Table 5.17. getNetworkStatus: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
coordinates	Coordinates	GPS coordinates in WGS84 / DMS0 format. Examples : N48d48m41s0.0 - E2d19m39s0.0 N45d0m14s0.0 - W1d11m49s0.0	1..1

## 8.3. Output parameter

**Table 5.18. getNetworkStatus: output parameters.**

Name	Type	Description	Cardinality
response	GetNetworkStatusResponse	A Network Stat Status telling whether the response is OK or not ; an Area Coverage array describing for each technology (2G, 3G, ...) if the network is covered or not, with a brief description ; an Breakdown Report array telling for each kind of network (2G Voice, 2G Data, 3G voice, ...) the kind of incident occurring, with a brief description.	1..1

## 8.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 8.5. Sample code

This sample shows you how Get a Network Status from a GPS coordinates data.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetNetworkStatusResponse;
import com.orange.api.malima.generated.service.networkkstatus.types.AreaCoverage;
import com.orange.api.malima.generated.service.networkkstatus.types.BreakdownReport;
import com.orange.api.malima.generated.types.Coordinates;

public class SdkGetNetworkStatus {

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
        }
    }
}

```

```

Malima.loadFromConfigFile(pathToConfigFile);

// create the coordinates you want to see
// here an example with the Eiffel Tower :
// http://maps.google.fr/?ie=UTF8&ll=48.858052,2.294469&spn=0.006847,0.018024&t=h&z=16
String latitude = "N48d51m28s9.0"; // 48.858052 North
String longitude = "E2d17m40s1.0"; // 2.294469 East

Coordinates coordinates = new Coordinates(latitude, longitude);

// Then call the service with requested Coordinates
GetNetworkStatusResponse response = null;
response = Malima.getNetworkStatus(malimaUser, coordinates);

StringBuffer result = new StringBuffer();
if (response != null) {
    if (response.getAreaCoverage() != null) {
        result.append("Area coverage      : \n");
        for (AreaCoverage areaCoverage : response.getAreaCoverage()) {
            if (areaCoverage.getTechnology() != null) {
                result.append("  Area technology      : "
                    + areaCoverage.getTechnology().getValue() + "\n");
            } else {
                result.append("  Area technology      : unknown\n");
            }
            result.append("  Area available      : ");
            if (areaCoverage.isCoverage()) {
                result.append("yes\n");
            } else {
                result.append("no\n");
            }
            result.append("  ---\n");
        }
    } else {
        result.append("Area coverage      : unknown\n");
    }
    result.append("---\n");
    if (response.getAreaCoverageDescription() != null) {
        result.append("Area coverage descr. : \n");
        for (String areaCoverageDesc : response.getAreaCoverageDescription()) {
            result.append("  Area description    : " + areaCoverageDesc + "\n");
            result.append("  ---\n");
        }
    } else {
        result.append("Area coverage descr. : unknown\n");
    }
    result.append("---\n");
    if (response.getBreakdownReport() != null) {
        result.append("Breakdown report : \n");
        for (BreakdownReport breakdownReport : response.getBreakdownReport()) {
            if (breakdownReport.getCommunicationType() != null) {
                result.append("  Communication type : "
                    + breakdownReport.getCommunicationType().getValue() + "\n");
            } else {
                result.append("  Communication type : unknown\n");
            }
            if (breakdownReport.getClassification() != null) {
                result.append("  Classification     : "
                    + breakdownReport.getClassification().getValue() + "\n");
            } else {

```

```

        result.append("  Classification      : unknown\n");
    }
    result.append("  ---\n");
}
} else {
    result.append("Breakdown report : unknown\n");
}
result.append("----\n");
} else {
    result.append("No response returned\n");
}
}
System.out.println("GetNetworkStatus response for Coordinates [ "
    + latitude + " - " + longitude + " ] : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The output looks like:

```

GetNetworkStatus response for Coordinates [N48d51m28s9.0 - E2d17m40s1.0] :
Area coverage      :
  Area technology   : TECH_2G
  Area available    : yes
  ---
  Area technology   : TECH_3G_2100
  Area available    : no
  ---
  Area technology   : TECH_3G_900
  Area available    : no
  ---
---
Area coverage descr. :
  Area description  : 2G EDGE
  ---
---
Breakdown report :
  Communication type : DATA_3G
  Classification      : NO_INFORMATION_AVAILABLE
  ---
  Communication type : VOICE_3G
  Classification      : MINOR_INCIDENT
  ---
  Communication type : VOICE_2G
  Classification      : MAJOR_INCIDENT
  ---
  Communication type : DATA_2G
  Classification      : NO_INCIDENT
  ---
---

```

## 9. GetIncidentDiagnostics method

### 9.1. Description

Get an Incident diagnostics information concerning a Line Identifier.

### 9.2. Input parameter

**Table 5.19. getIncidentDiagnostics: Input parameters.**

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
lineIdentifier	LineIdentifier	The SIM card identifier referenced by a subscriptionNumber, a simSerialNumber, a deviceImei, a deviceSerialNumber, a deviceUniqueId, a machineSerialNumber, or a phone number associated to voice, data or fax.	1..1

### 9.3. Output parameter

**Table 5.20. getIncidentDiagnostics: output parameters.**

Name	Type	Description	Cardinality
response	GetIncidentDiagnosticsResponse	An IncidentDiagnostics response for the given LineIdentifier concerning geolocation data of the SIM card, the last known geolocation area of the SIM card, and the network status concerning the SIM card area.	1..1

### 9.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

### 9.5. Sample code

This sample shows you how get an incident diagnostics result from a line identifier.

```

package com.orange.api.malima.doc.sdkGuide;

import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetIncidentDiagnosticsResponse;
import com.orange.api.malima.generated.service.networkstatus.types.AreaCoverage;
import com.orange.api.malima.generated.service.networkstatus.types.BreakdownReport;
import com.orange.api.malima.generated.types.LineIdentifier;

public class SdkGetIncidentDiagnostics {

```

```

private static String yourSubscriptionNumber = "your_subscription_number";

public static void main(String[] args) {
    try {
        // Create and initialize a malimaUser for Malima
        // Service from specified configuration
        String pathToConfigFile = "your/path/to/configFile.properties";
        MalimaUser malimaUser = new MalimaUser();
        malimaUser.loadFromConfigFile(pathToConfigFile);
        Malima.loadFromConfigFile(pathToConfigFile);

        // create the LineIdentifier you want to see
        LineIdentifier lineIdentifier = new LineIdentifier();
        // set the subscription number of the LineIdentifier
        lineIdentifier.setSubscriptionNumber(yourSubscriptionNumber);
        // Then call the service with requested LineIdentifier
        GetIncidentDiagnosticsResponse response = null;
        response = Malima.getIncidentDiagnostics(malimaUser, lineIdentifier);

        StringBuffer result = new StringBuffer();
        if (response != null) {
            if (response.getIncidentDiagnosticsStatus() != null) {
                result.append("Incident diag. status : "
                    + response.getIncidentDiagnosticsStatus().getValue() + "\n");
            } else {
                result.append("Incident diag. status : unknown\n");
            }
        }
        result.append("---\n");
        if (response.getGeolocationInformation() != null) {
            if (response.getGeolocationInformation().getGeolocationDate() != null) {
                result.append("Geolocation date : "
                    + response.getGeolocationInformation().getGeolocationDate()
                    .toString() + "\n");
            } else {
                result.append("Geolocation date : unknown\n");
            }
        } else {
            result.append("Geolocation info. : unknown\n");
        }
        result.append("---\n");
        if (response.getNetworkStatusResponse() != null) {
            result.append("Network status : \n");
            if (response.getNetworkStatusResponse().getStatus() != null) {
                result.append("Network stat status : "
                    + response.getNetworkStatusResponse().getStatus().getValue()
                    + "\n");
            } else {
                result.append("Network stat status : unknown\n");
            }
        }
        if (response.getNetworkStatusResponse().getAreaCoverage() != null) {
            result.append("Area coverage : \n");
            for (AreaCoverage areaCoverage : response
                .getNetworkStatusResponse().getAreaCoverage()) {
                if (areaCoverage.getTechnology() != null) {
                    result.append(" Area technology : "
                        + areaCoverage.getTechnology().getValue() + "\n");
                } else {
                    result.append(" Area technology : unknown\n");
                }
            }
            result.append(" Area available : ");
        }
    }
}

```

```

        if (areaCoverage.isCoverage()) {
            result.append("yes");
        } else {
            result.append("no");
        }
        result.append("\n");
        result.append("  ---\n");
    }
} else {
    result.append("Area coverage          : unknown\n");
}
result.append("----\n");
if (response.getNetworkStatusResponse().getAreaCoverageDescription() != null) {
    result.append("Area coverage descr.  : \n");
    for (String areaCoverageDesc : response.getNetworkStatusResponse()
        .getAreaCoverageDescription()) {
        result.append("  Area description    : " + areaCoverageDesc
            + "\n");
        result.append("  ---\n");
    }
} else {
    result.append("Area coverage descr.  : unknown\n");
}
result.append("----\n");
if (response.getNetworkStatusResponse().getBreakdownReport() != null) {
    result.append("Breakdown report : \n");
    for (BreakdownReport breakdownReport : response
        .getNetworkStatusResponse().getBreakdownReport()) {
        if (breakdownReport.getClassification() != null) {
            result.append("  Classification      : "
                + breakdownReport.getClassification().getValue() + "\n");
        } else {
            result.append("  Classification      : unknown\n");
        }
        if (breakdownReport.getCommunicationType() != null) {
            result.append("  Communication type  : "
                + breakdownReport.getCommunicationType().getValue() + "\n");
        } else {
            result.append("  Communication type  : unknown\n");
        }
        result.append("  ---\n");
    }
} else {
    result.append("Breakdown report : unknown\n");
}
result.append("----\n");
} else {
    result.append("Network status          : unknown\n");
}
}

} else {
    result.append("No response returned\n");
}
System.out
    .println("GetIncidentDiagnostics response for Subscription Number "
        + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {

```

```

functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}

```

The output looks like:

```

GetIncidentDiagnostics response for Subscription Number 23697260 :
Incident diag. status : NO_GEOLOCATION_AVAILABLE
---
Geolocation date      : unknown
---
Network status       : unknown

```

## 10. SubmitSessionHistory and GetSessionHistory methods

### 10.1. SubmitSessionHistory method

#### 10.1.1. Description

Submit a request for a given SIM card in order to get all traffic info occurred in a period.

#### Tip

The method `SubmitSessionHistory` is asynchronous and has to be used prior the `getSessionHistory` method call which gets back the submit result.

#### 10.1.2. Input parameter

Table 5.21. `submitSessionHistory`: Input parameters.

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
lineIdentifier	LineIdentifier	A unique Line Identifier	1..1
startDate	Calendar	The start date of the request	1..1
endDate	Calendar	The end date of the request	1..1

#### 10.1.3. Output parameter

Table 5.22. `submitSessionHistory`: output parameters.

Name	Type	Description	Cardinality
ticketNumber	int	A ticket number usable in the <code>getSessionHistory</code> request.	1..1

## 10.1.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 10.2. GetSessionHistory method

### 10.2.1. Description

Get the result of a `SubmitSessionHistory` method call.

#### Tip

The method `GetSessionHistory` has to be used after a `SubmitSessionHistory` method call, in order to get the asynchronous submit result.

As the method is the result of a preliminar asynchronous submit call, the result of the get method contains a `Ticket Status` telling wether the submit call work is finished, in progress or not started yet.

### 10.2.2. Input parameter

Table 5.23. `getSessionHistory`: Input parameters.

Name	Type	Description	Cardinality
malimaUser	MalimaUser	A Configurator for the Malima service	1..1
ticketNumber	int	A ticket number returned by the <code>submitSessionHistory</code> method call.	1..1

### 10.2.3. Output parameter

Table 5.24. `getSessionHistory`: output parameters.

Name	Type	Description	Cardinality
response	<code>GetSessionHistoryResultResponse</code>	A global ticket status telling wether the request is being processed, in progress or terminated ; a Billing Account data ; a Line data (referring an array of <code>CustomerData</code> , a <code>Subscription</code> , a <code>SIM card</code> , a <code>Device</code> , a machine identifier) ; and an array of <code>Session</code> .	1..1

### 10.2.4. Exceptions

This web method may throw `TechnicalException` or `FunctionalException`.

## 10.3. Sample code

This sample shows how to Submit then Get Session History of a SIM card in a period.

```
package com.orange.api.malima.doc.sdkGuide;
import java.util.Calendar;
```

```

import java.util.GregorianCalendar;
import com.orange.api.common.exception.ConfigurationException;
import com.orange.api.common.exception.FunctionalException;
import com.orange.api.common.exception.TechnicalException;
import com.orange.api.malima.Malima;
import com.orange.api.malima.MalimaUser;
import com.orange.api.malima.generated.service.GetSessionHistoryResponse;
import com.orange.api.malima.generated.service.sessionhistory.types.Session;
import com.orange.api.malima.generated.types.LineIdentifier;
import com.orange.api.malima.generated.types.TicketStatusType;

public class SdkSubmitAndGetSessionHistory {
    private static String yourSubscriptionNumber = "your_subscription_number";

    public static void main(String[] args) {
        try {
            // Create and initialize a malimaUser for Malima
            // Service from specified configuration
            String pathToConfigFile = "your/path/to/configFile.properties";
            MalimaUser malimaUser = new MalimaUser();
            malimaUser.loadFromConfigFile(pathToConfigFile);
            Malima.loadFromConfigFile(pathToConfigFile);

            /*****
            /** call SubmitSessionHistory method **/
            *****/
            // create the LineIdentifier you want to see history
            LineIdentifier lineIdentifier = new LineIdentifier();
            // set the subscription number of the LineIdentifier
            lineIdentifier.setSubscriptionNumber(yourSubscriptionNumber);
            // set the starting date and ending date of the session history
            Calendar startDateTime = new GregorianCalendar();
            Calendar endDateTime = new GregorianCalendar();
            startDateTime.set(2010, 0, 1); // jan, 1st 2010
            endDateTime.set(2010, 1, 1); // feb, 1st 2010

            StringBuffer result = new StringBuffer();
            result.append("==> SUBMIT METHOD CALL :\n");
            // Then call the service with requested LineIdentifier and period
            int ticketNumber = -1;
            ticketNumber = Malima.submitSessionHistory(malimaUser, lineIdentifier,
                startDateTime, endDateTime);

            if (ticketNumber > -1) {
                result.append("Ticket number returned : " + ticketNumber);

                /*****
                /** call GetSessionHistory method **/
                *****/
                result.append("\n");
                result.append("====> GET METHOD CALL :\n");
                // if there is a significant response, the GetSessionHistory method will
                // be called
                GetSessionHistoryResponse getResponse = null;
                TicketStatusType ticketStatus = TicketStatusType.WAITING;

                // get session history according to the ticket number return by the
                // submit method
                while ((!ticketStatus.equals(TicketStatusType.TERMINATED))
                    && (!ticketStatus.equals(TicketStatusType.ERROR))) {

```

```

getResponse = Malima.getSessionHistory(malimaUser, ticketNumber);
// you could wait for a while here ...
try{
    Thread.currentThread().sleep(5000);//sleep for 5 seconds
}
catch(InterruptedException ie){
    result.append("WARN : GetSessionHistory interrupted while sleeping 5 secs" );
}
if (getResponse != null) {
    ticketStatus = getResponse.getGlobalProcessingStatus();
} else {
    ticketStatus = TicketStatusType.ERROR;
}
} // while get not terminated
result.append("Ticket Status : " + ticketStatus + "\n");
// there is a response, check if the response is fine
if (getResponse != null
    && getResponse.getGlobalProcessingStatus() != null
    && getResponse.getGlobalProcessingStatus().getValue().equals(
        TicketStatusType.TERMINATED)) {
result.append("Process value returned : "
    + getResponse.getGlobalProcessingStatus().getValue() + "\n");
if (getResponse.getLine() != null) {
    result.append("----\n");
    result.append("Machine identifier      : "
        + getResponse.getLine().getMachineIdentifier() + "\n");
}
if (getResponse.getBillingAccount() != null) {
    result.append("----\n");
    result.append("Billing Account      : "
        + getResponse.getBillingAccount().getBillingAccountNumber()
        + "\n");
    if (getResponse.getBillingAccount().getBillingAddress() != null) {
        result.append("Billing Account Address: "
            + getResponse.getBillingAccount().getBillingAddress()
            .getStreetNameAndNumber()
            + " - "
            + getResponse.getBillingAccount().getBillingAddress()
            .getTown() + "\n");
    }
} else {
    result.append("----\n");
    result.append("No billing acct found : \n");
}
if (getResponse.getSessions() != null
    && getResponse.getSessions().getSession() != null) {
    result.append("----\n");
    result.append("Sessions              : \n");
    for (Session session : getResponse.getSessions().getSession()) {
        result.append(" Terminal Id          : "
            + session.getTerminalId() + "\n");
        result.append(" Country               : " + session.getCountry()
            + "\n");
        if (session.getBearerType() != null) {
            result.append(" Terminal Id          : "
                + session.getBearerType().getValue() + "\n");
        }
        if (session.getDurationInSeconds() != null) {
            result.append(" Duration (sec)       : "
                + session.getDurationInSeconds().toString() + "\n");
        }
    }
}

```

```

    }
    if (session.getVolumeOutInKB() != null) {
        result.append("  Volume (kbs)           : "
            + session.getVolumeOutInKB().toString() + "\n");
    }
    result.append("  ----\n");
}
} else {
    result.append("----\n");
    result.append("No sessions found.\n");
}
} else {
    result
        .append("No session history result found.\n");
}
} else {
    result.append("No response returned\n");
}
}

System.out.println("SessionHistory response for Subscription Number "
    + yourSubscriptionNumber + " : \n" + result.toString());

} catch (ConfigurationException configurationException) {
    configurationException.printStackTrace();
} catch (FunctionalException functionalException) {
    functionalException.printStackTrace();
} catch (TechnicalException technicalException) {
    technicalException.printStackTrace();
}
}
}

```

The output looks like:

```

SessionHistory response for Subscription Number 23697253 :
==> SUBMIT METHOD CALL :
Ticket number returned : 573
=====> GET METHOD CALL :
Ticket Status : TERMINATED
No session history result found.

```

# Appendix A. Error codes

## Caution

The error message returned by the M2M API may contain a reference to <user name>. It refers to the couple Access Key - Access Key Password used for authentication.

**Table A.1. Error codes**

Code	Detail
malimaErrorFault	<p>Error message: Une erreur Malima est survenue..</p> <p>Can be of several kinds:</p> <p>MLM_UNKNOWN_USER : L'utilisateur &lt;user name&gt; est inconnu de Malima.</p> <p>Input parameter: <i>user name</i>.</p> <p>CANNOT_CONNECT_TO_DATABASE : Accès impossible à la base de donnée.</p> <p>MLM_TICKET_CREATION : La création du ticket a échoué.</p> <p>MLM_NETWORKSTATUS_SERVICE_CONNECTION_ERROR : Impossible de se connecter au service de météo du réseau.</p> <p>MLM_NETWORKSTATUS_SERVICE_ACCESS_DENIED : Accès au service de météo du réseau non autorisé.</p> <p>MLM_GEOLOC_SERVICE_CONNECTION_ERROR : Impossible de se connecter au service de géolocalisation.</p> <p>MLM_GEOLOC_SERVICE_ACCESS_DENIED : Accès au service de géolocalisation non autorisé.</p>
unknownLineIdentifier ErrorFault	<p>Error message: Unknwon line identifier / Identifiant de ligne inconnu..</p> <p>Description: choose only one of the following element:</p> <p>subscriptionNumber, simSerialNumber, deviceImei, deviceSerialNumber, deviceUniqueId, machineSerialNumber, msisdnVoice, msisdnData, msisdnFax.</p>
tooManyLine IdentifiersFault	<p>Error message: Too many lines identifiers.</p> <p>Description: Le nombre d'identifiants de ligne est trop élevé (&lt;maximum lines number&gt; au maximum).</p> <p>Input parameter: <i>maxAllowedLineIdentifiersNumber</i>.</p>

Code	Detail
invalidParameterFault	Error message: At least one parameter is not valid. Description: Au moins un paramètre n'est pas valide.
unknownTicketFault	Error message: Unknown ticket number. Description: Le ticket <ticket number> est inconnu. Input parameter: <i>unknownTicketNumber</i>
resultAlreadyRetrievedFault	Error message: If a user try to retrieve a result he has already retrieved. Description: Le résultat a déjà été récupéré le <fisrt retrieval date>. Input parameter: <i>retrievalDate</i>
networkStatusServiceErrorFault	Error message: networkStatusServiceErrorFault : Network status IS error. Description: Une erreur est survenue dans le service de météo du réseau.
invalidPeriodErrorFault	Error message: Invalid period error. Description: Période de recherche non valide. Malima error messages listing : La date de fin est antérieure à la date de début. La date de début de recherche remonte à plus d'un an. La durée de la période de recherche excède 30 jours.

More specific error can be returned into M2M API responses, they are returned into M2M tickets responses when requesting asynchronous (i.e. submitXxx, then getXxx requests) queries:

**Table A.2. Ticket Error codes**

Code	Detail
MLM_TELCO_IS_ERROR	[0000000573] Un seul critère de positionnement doit être renseigné.
MLM_TELCO_IS_ERROR	[0000000574] Un critère de positionnement doit être renseigné.
MLM_TELCO_IS_ERROR	[0000000575] Le N° d'abonnement est inconnu.
MLM_TELCO_IS_ERROR	[0000000581] Non autorisé à modifier l'abonnement.
MLM_TELCO_IS_ERROR	[0000000584] La date d'effet doit être supérieure ou égale à la date du jour.
MLM_TELCO_IS_ERROR	[0000000585] Le Code service est inconnu.

M2M API for Java - Error codes

Code	Detail
MLM_TELCO_IS_ERROR	[0000000586] Le code service est inconnu
MLM_TELCO_IS_ERROR	[0000000597] Le code service a une date d'effet inférieure à la date de connexion.
MLM_TELCO_IS_ERROR	[0000000598] Le code service est recurrent et existe déjà sur l'abonné.
MLM_TELCO_IS_ERROR	[0000000600] La date d effet est invalide elle doit être au premier du mois.
MLM_TELCO_IS_ERROR	[0000000655] La zone numérique passée en paramètre n'est pas valide.
MLM_TELCO_IS_ERROR	[0000000675] Aucun abonné n'est éligible.
MLM_TELCO_IS_ERROR	[0000000711] Positionnement impossible le service existe déjà sur l'abonné.
MLM_TELCO_IS_ERROR	[0000000794] La zone numerique passée en parametre n'est pas valide.
MLM_TELCO_IS_ERROR	[0000000795] La date passée en parametre n'est pas valide.
MLM_TELCO_IS_ERROR	[0000000796] Le N° d'abonnement est obligatoire.
MLM_TELCO_IS_ERROR	[0000000800] Le N° d'abonnement est inconnu.
MLM_TELCO_IS_ERROR	[0000000842] Le nombre d'abonnés dépasse le plafond.
MLM_TELCO_IS_ERROR	[0000000844] La zone numérique passée en paramètre n'est pas valide.
MLM_TELCO_IS_ERROR	[0000000848] Le nombre d'itérations demandées dépasse le seuil max. de restitution.
MLM_TELCO_IS_ERROR	[0000000987] Un seul critère de positionnement doit être renseigné.
MLM_TELCO_IS_ERROR	[0000000988] Un critère de positionnement doit être renseigné.
MLM_TELCO_IS_ERROR	[0000000989] Le N° d'abonnement est inconnu.
MLM_TELCO_IS_ERROR	[0000000995] Non autorisé à modifier l'abonnement.
MLM_TELCO_IS_ERROR	[0000000996] Renseigner code option ou code type de modification.
MLM_TELCO_IS_ERROR	[0000000997] Le code modification est obligatoire.
MLM_TELCO_IS_ERROR	[0000000998] Code option invalide.
MLM_TELCO_IS_ERROR	[0000000999] Code type de modification et/ou Code modification invalides.
MLM_TELCO_IS_ERROR	[0000001000] Type de suspension obligatoire.
MLM_TELCO_IS_ERROR	[0000001001] N° d'abonnement obligatoire.
MLM_TELCO_IS_ERROR	[0000001008] Attention, abonné déconnecté ou en attente de résiliation.

M2M API for Java - Error codes

Code	Detail
MLM_TELCO_IS_ERROR	[0000001009] Attention, abonné non connecté ou en attente de connexion.
MLM_TELCO_IS_ERROR	[0000001010] Attention, abonné suspendu ou en attente de suspension.
MLM_TELCO_IS_ERROR	[0000001012] Demande incompatible avec le tarif de l'abonné.
MLM_TELCO_IS_ERROR	[0000001013] Service incompatible avec l'option demandée
MLM_TELCO_IS_ERROR	[0000001019] Migration non autorisée.
MLM_TELCO_IS_ERROR	[0000001023] Date d'effet incorrecte.
MLM_TELCO_IS_ERROR	[0000001024] Type de suspension incorrect.
MLM_TELCO_IS_ERROR	[0000001025] Demande non applicable.
MLM_TELCO_IS_ERROR	[0000001026] Contact inconnu.
MLM_TELCO_IS_ERROR	[0000001028] Demande non applicable.
MLM_TELCO_IS_ERROR	[0000001035] N° abonnement obligatoire.
MLM_TELCO_IS_ERROR	[0000001037] Date d'effet incorrecte.
MLM_TELCO_IS_ERROR	[0000001038] Demande non applicable.
MLM_TELCO_IS_ERROR	[0000001039] Valeur non autorisée.
MLM_TELCO_IS_ERROR	[0000001040] L'événement n'a pas été annulé.
MLM_TELCO_IS_ERROR	[0000001044] Demande non applicable.
MLM_TELCO_IS_ERROR	[0000001045] Aucune connexion en attente.
MLM_TELCO_IS_ERROR	[0000001046] Abonné sans suspension programmée.
MLM_TELCO_IS_ERROR	[0000001047] Abonné sans résiliation programmée.
MLM_TELCO_IS_ERROR	[0000001048] Abonné sans rétablissement programmé.
MLM_TELCO_IS_ERROR	[0000001049] Abonné sans activation programmée.
MLM_TELCO_IS_ERROR	[0000001095] L'abonné doit être actif pour ce service.
MLM_TELCO_IS_ERROR	[0000001097] Le Numéro de Carte SIM (NSCE) n'existe pas.
MLM_TELCO_IS_ERROR	[0000001098] Numéro de Carte SIM 2 (NSCE) non renseigné.
MLM_TELCO_IS_ERROR	[0000001101] Le Numéro de Carte SIM 2 (NSCE) n'est pas associé à l'abonnement.
MLM_TELCO_IS_ERROR	[0000001102] Le Service doit être positionné sur l'abonnement.
MLM_TELCO_IS_ERROR	[0000001103] Paramètre d'entrée 'Type de carte' incorrect.
MLM_TELCO_IS_ERROR	[0000001104] Le service n'a pas pu être ajouté sur certains(s) abonné(s).
MLM_TELCO_IS_ERROR	[0000001105] Aucun service n'a été positionné.
MLM_TELCO_IS_ERROR	[0000001107] L'abonné possède déjà une carte de renouvellement en cours.

M2M API for Java - Error codes

Code	Detail
MLM_TELCO_IS_ERROR	[0000001108] L'abonné ne possède pas de carte de ce type active.
MLM_TELCO_IS_ERROR	[0000001122] La valeur du paramètre Type de suspension est incorrecte.
MLM_TELCO_IS_ERROR	[0000001123] L'une des dates passée en paramètre est invalide.
MLM_TELCO_IS_ERROR	[0000001126] Date d'effet incorrecte.
MLM_TELCO_IS_ERROR	[0000001129] Code modification inexistant ou inactif.
MLM_TELCO_IS_ERROR	[0000001132] Demande non applicable.
MLM_TELCO_IS_ERROR	[0000001269] Le N° d'abonnement doit être renseignés
MLM_TELCO_IS_ERROR	[0000001270] L'état de la ligne ne permet pas sa suspension
MLM_TELCO_IS_ERROR	[0000001275] Impossible de suspendre un abonné qui va être résilié
MLM_TELCO_IS_ERROR	[0000001277] Cette option n'est pas valable pour une ligne terminée
MLM_TELCO_IS_ERROR	[0000001279] On ne peut rétablir qu'un abonné déjà suspendu
MLM_TELCO_IS_ERROR	[0000001280] Cette option n'est pas valable pour une ligne terminée
MLM_TELCO_IS_ERROR	[0000001282] Cette option n'est pas disponible pour une ligne principale
MLM_TELCO_IS_ERROR	[0000001284] Cette option n'est pas valable pour une ligne terminée
MLM_TELCO_IS_ERROR	[0000001286] Cette option n'est pas disponible pour une ligne principale
MLM_TELCO_IS_ERROR	[0000001325] Demande non applicable
MLM_TELCO_IS_ERROR	[0000001326] Demande incompatible avec tarif de l'abonné.
MLM_TELCO_IS_ERROR	[0000001327] Service incompatible avec l'option demandée.
MLM_TELCO_IS_ERROR	[0000001328] Date effet obligatoire.
MLM_TELCO_IS_ERROR	[0000001334] Résiliation impossible.
MLM_TELCO_IS_ERROR	[0000001335] Résiliation impossible
MLM_TELCO_IS_ERROR	[0000001357] N° abonnement obligatoire
MLM_TELCO_IS_ERROR	[0000001360] Date d'effet incorrecte.
MLM_TELCO_IS_ERROR	[0000001364] Le N° d'abonnement est obligatoire.
MLM_TELCO_IS_ERROR	[0000001366] la date de début des appels à rechercher est obligatoire.
MLM_TELCO_IS_ERROR	[0000001367] Le N° d'abonnement est inconnu.

M2M API for Java - Error codes

Code	Detail
MLM_TELCO_IS_ERROR	[0000001368] Le nombre d'itérations demandées dépasse le seuil.
MLM_TELCO_IS_ERROR	[0000001369] Le format de la date de début est incorrect.
MLM_TELCO_IS_ERROR	[0000001370] Le format de la date de fin est incorrect.
MLM_TELCO_IS_ERROR	[0000001371] Le format de l'heure de début est incorrect.
MLM_TELCO_IS_ERROR	[0000001372] Le format de l'heure de fin est incorrect.
MLM_TELCO_IS_ERROR	[0000001375] Date & heure de fin de recherche inférieure à date & heure de début.
MLM_TELCO_IS_ERROR	[0000002000] Action impossible : cet abonné est suspendu/déconnecté.
MLM_TELCO_IS_ERROR	[0000002001] Cet ajout de service porte sur un abonné suspendu/déconnecté.
MLM_TELCO_IS_ERROR	[0000002008] Ne pas renseigner Type numéro et Numéro si N° d'abonnement renseigné
MLM_TELCO_IS_ERROR	[0000002009] Ne pas renseigner N° d'abonnement si Type numéro et Numéro renseignés
MLM_TELCO_IS_ERROR	[0000002010] Numéro doit être renseigné si Type numéro est renseigné
MLM_TELCO_IS_ERROR	[0000002011] Type de numéro doit être renseigné si Numéro est renseigné.
MLM_TELCO_IS_ERROR	[0000002012] Le type de numéro n'est pas valide.
MLM_TELCO_IS_ERROR	[0000002013] Le couple Type et Numéro DISE n'est pas valide
MLM_TELCO_IS_ERROR	[0000002014] N° abonnement ou Type de numéro/Numéro ne sont pas renseignés
MLM_TELCO_IS_ERROR	Opération non réalisée. Veuillez vous rapprocher du service assistance client au {0} pour plus de précisions.
UPDATE_IN_PROGRESS	Une requête de mise à jour du statut de la SIM est déjà en cours pour cet abonnement.
STATUS_NOT_ALLOWED	Le nouveau statut demandé (<requested status>) n'est pas compatible avec le statut courant (<current status>).
UPDATE_SIM_SUSPENDED_NOT_BY_CUSTOMER	Opération interdite car la SIM n'a pas été suspendue par le client.
SIM_NOT_ENOUGH_IN_ACTIVATION	La SIM doit avoir été activée pendant <duration> mois avant de pouvoir passer dans l'état activée pour test.
UPDATE_SIM_STATUS_SLEEPING	La SIM sera activée à la première communication.
MLM_TELCO_IS_CONNECTION_ERROR	Impossible de se connecter à DISE (<error details>).
MLM_INTERNAL_ERROR	Une erreur interne Malima est survenue.

Code	Detail
MLM_NETWORK_STATUS_ERROR	Erreur Requête mal-formée
MLM_NETWORK_STATUS_ERROR	Format des coordonnées incorrect
MLM_NETWORK_STATUS_ERROR	Coordonnées XY ne correspondent pas à des coordonnées en France
MLM_NETWORK_STATUS_ERROR	Interface Network Status indisponible

## 1. Connectivity

Table A.3. Connectivity errors

Code	Detail
REQ.BACKSIDE_CONNECTION_FAILURE	Failed to establish a backside connection with the back-end service  (500) Network connection error between technical platform and WSP server.
GENERAL	Internal Error  (500) SOAP message specify an unknown operation name like "operation_C" or SOAP message has a missing '>' for the closing tag soapenv:Envelope.
REQ.SERVICE_NOT_FOUND	The back-end service could not handle the request because it has not found any service corresponding to the received request  (500) Service is not available.
IOSW.UNKNOWN_ERROR	An unknown error has occurred during the processing of the request/response  (500) « Internal error » of the technical platform.
REQ.AUTHENTICATION_FAILURE	The credentials provided are not valid  (401) Unauthorized.
REQ.AUTHORISATION_FAILURE	The client is not authorized to consume the requested service  (500).
REQ.SERVICE_AUTHENTICATION_FAILURE	The credentials provided by the service broker when connecting to the back-end service are not valid  (500).

M2M API for Java - Error codes

<b>Code</b>	<b>Detail</b>
REQ.SERVICE_AUTORISATION_FAILURE	The service broker is not authorized to consume the requested back-end service  (500).
REQ.SCHEMA_VALIDATION_ERROR	The request sent by the client does not conform to the xml schema of the service  (500) Schema Validation Error.
IOSW.XML_THREAT_ERROR	An xml threat was detected by the technical validation platform  XML Threat protection, attacks, XDoS.
MESSAGE_TOO_LARGE	The size of the request or response message is too large  XML Threat protection - Message too large.
REQ.SERVICE_INTERNAL_ERROR	The service failed to respond to the request because of an internal error  Server Internal error.