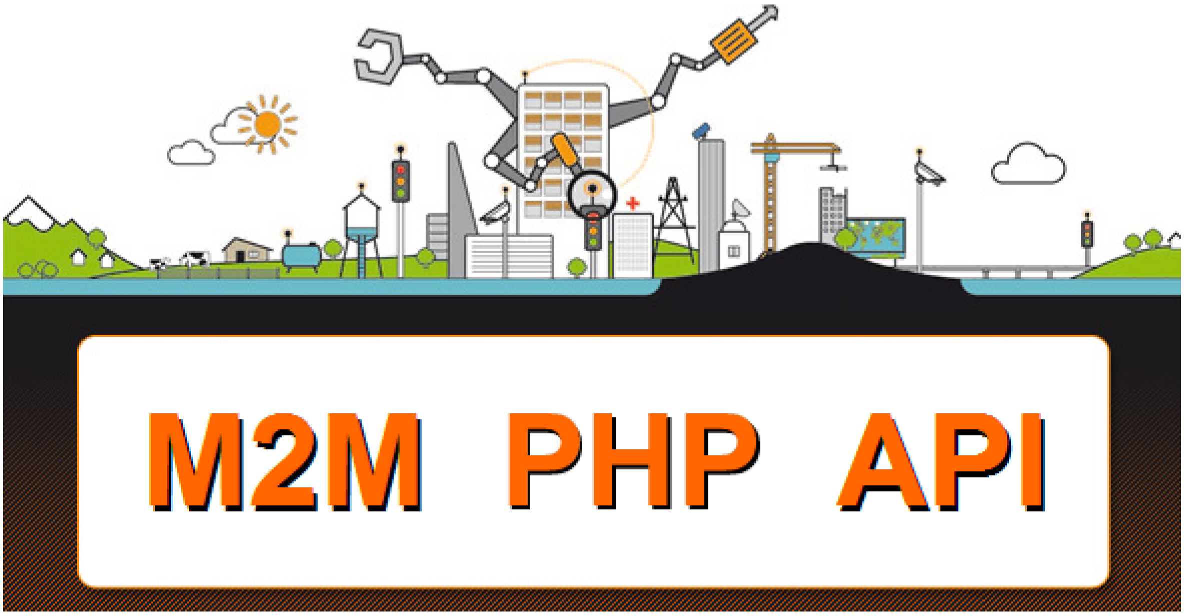
SDK MANUAL



M2M PHP API

Version 3.20.0

beginner

Revision History

Revision 3.20.0	2015/01/22
• Business rules: this feature links the alarm triggers with the sim card status management. • Obsolete SubscriptionStatus and IncidentDiagnostics V1 services were removed from the SDK.	
Revision 3.19.0	2014/10/29
• Session History, Traffic Tracking, Customer alarm : before version 3.19 the data communications were split between GPRS and UMTS bearers. Now, 2G and 3G data are all linked to the GPRS bearer. The UMTS bearer is no longer used.	
Revision 3.16.0	2014/08/14
• Connectivity Directory : new data (IMSI, ICCID, account name), new search criteria (user, IMSI) and new order criteria • Network Status : new information on the network coverage (breakdownReportDescription, networkResolutionMessage, areaCoverageCode, breakdownReportCode, forecastCoverage) • IncidentDiagnostics : new network status available for 3G 900, 3G 2100, H+, 4G	
Revision 3.11.2	2014/02/10
• New service Device Info : get real-time monitoring and location information for a set of sim cards. • ConnectivityDirectory evolution : display the last known location and sim card compatibility with the DeviceInfo service. • All services : pre authentication has been activated in order to avoid systematic Basic Auth "handshake". It improves response times when doing mass requests on a service. • Minor evolutions on documentation.	
Revision 3.10.1	2013/10/16
• CustomerAlarms : new trigger on subscription option change : an alarm will be raised when subscription options are added or deleted. • ConnectivityDirectory : new search based on IMSI in the getConnectivityDirectory method. • Minor bugs corrections and evolutions on code and documentation.	
Revision 3.9.0	2013/10/09
• Evolutions on ConnectivityDirectory (List of the services linked to the subscription. If the label of the service is not null, it's displayed, otherwise the service code is displayed). • CustomerAlarms : trigger on subscription status change : an alarm will be raised if M2M subscription status change (Sim status in Malima). • CustomerAlarms : trigger on "crazy" machine: an alarm will be raised if the cumulated traffic for a given call origin and bearer type exceeds the given threshold for the given period. • Minor bugs corrections and evolutions on code and documentation.	
Revision 3.7.0	2013/07/08
• New V2 version for customer alarms and triggers including alarms on "silent" machines, "lazy" machines, email notification. Sample code updates for "alarms" web service calls and notification. Minor bugs corrections and evolutions on code and documentation.	
Revision 2.2.0	2011/05/20
• Minor bugs corrections and evolutions on code and documentation.	

Version 3.20.0

Table of Contents

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.

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 Malima SDK. 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 all the 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

1.1. 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:

- ConnectivityDirectory : retrieve information (phone number, sim card status...) related to one or several sim cards in your fleet.
- ConnectivityDirectory : for each sim card, update your own data (identifier, name, description) for machine/ device in which the sim card is embedded.
- UpdateSimStatus : change the status (activate, deactivate...) of one or several sim cards in your fleet.
- TrafficTracking : get the aggregated value of the messages (local/roaming communications) exchanged by one or several machines / SIM cards for maximal period running from the first day of the current month to the current date.
- SessionHistory : list the communications (SMS, voice, data calls) made on a sim card for a given period of time.
- ServiceCustomerAlarm : manage triggers for an early detection (automatic alarms) of any incident related to your sim cards (over-consumption, unauthorized location, sim card moved to another device...)
- DeviceInfo : get monitoring and location information for a set of sim cards.
- NetworkStatus : get the Orange network covering status in a specific location defined by its latitude and longitude.
- IncidentDiagnostics : locate a specific sim card and check the Orange network coverage for this location.

1.2. Concepts

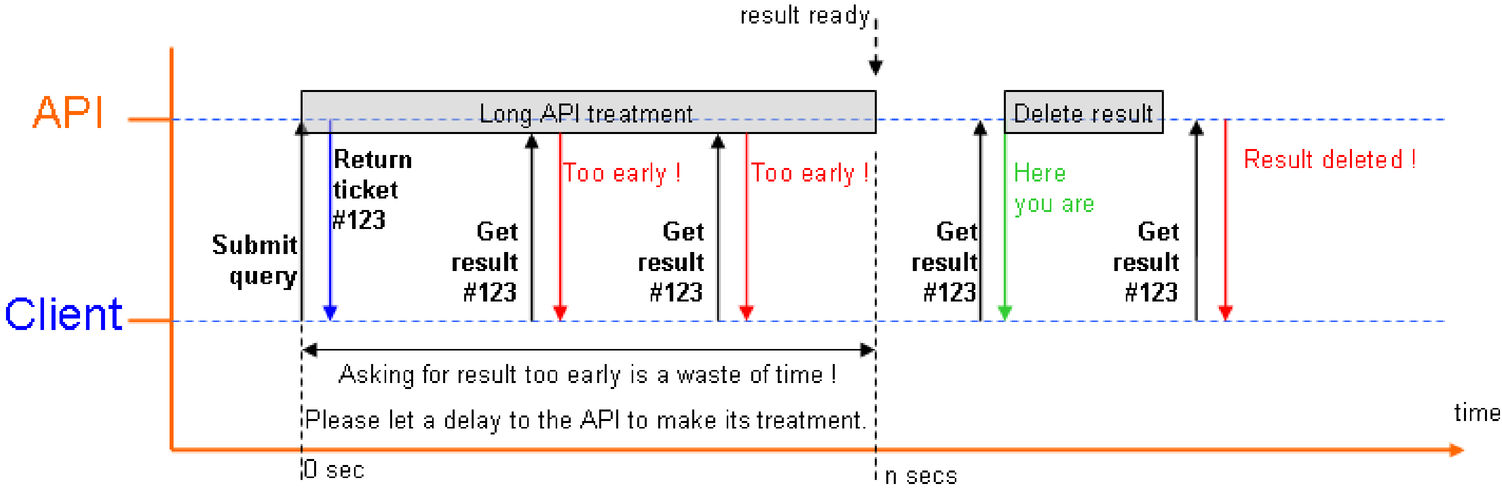
The provided machine-to-machine (M2M) services are web services included in a Software Development Toolkit, that can be inserted into software applications. They allow automating the deployment and management of your M2M sim cards fleet and early detection of incidents.

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

- The operator of the M2M machines, who expresses functional demands related to the management of his fleet.
- The developers who insert these services into the given application.

Please note that some operations are done in asynchronous mode because the process might last several minutes (example: sim card activation/deactivation). In this case, there is a two-step process: the end-user application submits its request (submitXXX call) which includes the input parameters. A ticket number is then returned to the customer by the Orange application. The end-user application makes getXXX calls (polling) with the ticket number as input parameter. The Orange application will return the current status for the ticket (WAITING, TERMINATED, ERROR). When the information requested by the end-user application is available, the getXXX call response has the TERMINATED status and contains the response data. 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 fair value for the delay between 2 getXXX calls would be 10 seconds for SessionHistory/TrafficTracking, 10 minutes for UpdateSimStatus. A small figure below explain how to query your result.

Figure 1.1. Example of an asynchronous sequence call with result retrieved later.



Chapter 2. What's new ?

If you are already familiar with the M2M SDK, you can focus on this chapter to get the latest features related to the SDK.

If you have never used the M2M SDK, you can skip this chapter and get more general information in the next chapters.

2.1. Business rules

This version provides a business rule allowing to link the trigger/alarm service with the sim lifecycle management service. When an alarm is raised for a sim card, the service can automatically modify the status of the sim card. For example, if a crazy machine alarm is triggered (over traffic on the device), the line can be automatically suspended, using the business rule.

```
 $oCreateTrigger= new createTrigger();
 //New scope
 $oScope = new scope();
 //Your subscription number have to be set in dataSet.php
 $oScope->lineIdentifier = new lineIdentifier();
 $oScope->lineIdentifier->subscriptionNumber=$sSubscriptionNumber;
 $oCreateTrigger->scope = $oScope;
 //New TriggerCriteria
 $oCreateTrigger->triggerCriteria = new TriggerCriteria();

 $oCreateTrigger->triggerCriteria->trafficCrazyMachineCriteria= new
TrafficCrazyMachineCriteria();

 $oCreateTrigger->triggerCriteria->trafficCrazyMachineCriteria->callOrigin=$sCallOrigin  ;
 $oCreateTrigger->triggerCriteria->trafficCrazyMachineCriteria->bearer=$sBearer2 ;
 $oCreateTrigger->triggerCriteria->trafficCrazyMachineCriteria->period=$iPeriod;
 $oCreateTrigger->triggerCriteria->trafficCrazyMachineCriteria->value=$iValue;
 $oCreateTrigger->comment =$sComment;
 $oCreateTrigger->level = $sLevel;
 $oCreateTrigger->notification =$bNotification;
 $oCreateTrigger->groupId = $iGroupId;

 $oBusinessRules = new businessRules();
 $oSlmBusinessRule = new slmBusinessRule();
 $oSlmBusinessRule->requestedStatus = "SUSPENDED";
 $oBusinessRules->slmBusinessRule = $oSlmBusinessRule;
 $oCreateTrigger->businessRules = $oBusinessRules;

 //Call
 $oCreateTriggerResponse=$oServiceCustomerAlarmClient->call_createTrigger($oCreateTrigger);
```

2.2. Subscription Status and IncidentDiagnosis V1 removed from the SDK.

- This SubscriptionStatus service is no longer used to refresh the data in the connectivity directory.

beginner

- This IncidentDiagnosis V1 service is no longer used for triggers and alarms. The V2 version should be used instead.

Chapter 3. Prerequisites

Before getting started, please read the information of this chapter, and follow any installation / download instructions. You'll be then ready to move onto the other chapters.

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

3.1. What you need to consume M2M API web service

To access the M2M API, you first need to get credentials (access key/password) from Orange. See this chapter for additional information. Once you have these items, please keep them, they will be necessary to access the M2M API.

A lot of examples require a subscription identifier for 1 sim card. It would be good to have one before starting using the SDK. If you do not have a subscription identifier, you can start by using the searchInConnectivityDirectory method, in order to retrieve the list of the sim cards and subscription identifiers for your customer account.

3.1.1. API access credentials needed

This section focuses on how to access the API and how to call the web services.

To use the API for your developments, you will need:

- *Service URLs*: the URLs to consume the web services
- *Access Key* and *Access Key Password* for authentication.

3.1.2. 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 shows the WSDL end-point locations depending on the M2M functionality and version you are working with.

Table 3.1. Which end-point should I use, and with which version of M2M API ?

Functionality	URL	Version
Connectivity Directory	*https://iosw-ba.orange.com:443/MLM/ConnectivityDirectory-1*	V1
Sim Lifecycle Management	*https://iosw-ba.orange.com:443/MLM/SimLifecycleManagement-1*	V1
Traffic Tracking	*https://iosw-ba.orange.com:443/MLM/TrafficTracking-1*	V1
Session History	*https://iosw-ba.orange.com:443/MLM/SessionHistory-2*	V2
Customer Alarm	*https://iosw-ba.orange.com:443/MLM/CustomerAlarm-2*	V2
Incident Diagnostics	*https://iosw-ba.orange.com:443/MLM/IncidentDiagnostics-2*	V2
Device Info	*https://iosw-ba.orange.com:443/MLM/DeviceInfo-1*	V1
Network Status	*https://iosw-ba.orange.com:443/MLM/NetworkStatus-1*	V1

3.2. PHP requirements

If you are going to use the SDK you can skip this section.

On the contrary, if you want to develop from scratch, you have to read this section.

You will need to download and to activate through `php.ini` the Soap, Curl and OpenSsl extension. The Zend Framework manages Soap objects and the logs.

Table 3.2. Required libraries

Library type	Library examples and comments
ZendFramework	The Zend Framework is required to use this sdk. It is provided with the sdk in the folder "COMMON\External\". The version used is ZendFramework-1.9.3PL1.
SOAP Library	We recommend to use Wamp server 2.2 [http://www.wampserver.com/download.php] which provides the Soap extension. You'll have to activate the extension through `php.ini`).
Curl Library	The recommended version is "libcurl/7.19.6 OpenSSL/0.9.8k zlib/1.2.3" which is included in Wamp server 2.2.You'll have to activate the extension through `php.ini`).
OpenSsl Library	The recommended version is OpenSSL 0.9.8k which is included in Wamp server 2.2.You'll have to activate the extension through `php.ini`).

Table 3.3. Optional libraries

Library type	Library examples and comments
Unit Testing	We recommend the use of PHP unit [http://www.phpunit.de/] or SimpleTest [http://simpletest.org/].
Logging	We recommend the use of Zend_Log [http://logging.apache.org/log4php/].

For the matter of simplicity and when the library license allows, we have included in our SDKs the above mentioned libraries.

3.3. PHP environment

We recommend the use of Eclipse PDT or Netbeans in association with xDebug as IDE to develop, debug and run your application. We encourage you as well to use the WAMP framework to test your application. The code has been tested and developped with the PHP5 version. You'll find the exact version number in the changelog of the SDK. So make sure to use the right PHP5 version to ensure compatibility.

To recap, here is what we recommend you to use:

Table 3.4. PHP environment recommended tools

Tool	Comment
Eclipse PDT IDE	PDT Eclipse IDE is freeware and provides developer with intellisense-like functionality. You may also use Netbeans with xDebug.
Wamp (WampServer Version 2.2)	Wamp (Windows Apache Mysql Php) allows you to switch easily between PHP and Apache versions.

Apache localhost instruction for Windows users: to modify the localhost directory (by default under Windows `c:/wamp/www` if you use Wampserver), find the `httpd.conf` file in the directory `wamp/bin/apache/apache.<version>/conf/` (where <version> is the version of Apache you are working with). Edit the following lines in the file:

`DocumentRoot "<path to your directory>"` and `Directory "<path to your directory>"`.

Save the file and restart Wamp.

Example:

```
DocumentRoot "c:/devarea"
<Directory "c:/devarea">
```

3.4. 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 eveything is working fine.

Chapter 4. Install and Set Up

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

4.1. Install or re-install M2M SDK in PHP

The SDK is composed of two libraries:

- The `Common` library and `M2M` (the core libraries),

These libraries are all mandatory.

```
<project>
- M2M
- COMMON
```

What you have to do now is rather simple if you followed previous document sections. Just copy the core libraries folders to your workspace and you are ready to go!

We encourage you to use Integrated Development Environment (IDE) which will ease your everyday work with programming. You can use Eclipse and Netbeans for free.

For M2M API users who have already installed a previous version of **M2M SDK**, they just have to delete old directories “COMMON” and “M2M”, and to copy new shipped ones.

The configuration file does not change, excepted when the M2M end-points change, for a new version of M2M for example.

4.2. One shot installation

You will learn here how to run the code snippets depending on your operating system.

4.3. First API call in PHP

If you reach this step it means that you are ready to deploy our SDK libraries and the SDK configuration file to get informations on a Sim card in PHP. This is what we will achieve here with this small sample demonstrating that you can successfuly communicate with our API by calling the `getConnectivityDirectory` method.

This `getConnectivityDirectory` method returns information (subscription number, status, MSISDN...) related to the selected sim card.

The first code listing below uses the configuration file `m2m.ini`. Do not forget to edit this file to type your own credentials, proxy and end-point data before running this example.

Here is an example of a sample code using the configuration file `m2m.ini`:

beginner

```
<?php
define("ABS_PATH", dirname(__FILE__));
//include datas required
include(ABS_PATH . "/resources/dataSet.php");
$sOutput = "";
$sInitializationPath = dirname ( __FILE__ ) . "/../M2M/Initialization.php";
if ( ! require_once ($sInitializationPath) ) {
throw new Exception ( "Cannot find Initialization file at $sInitializationPath" );
}
// Path to inifile
$oIniFile = new M2M_IniParser(dirname(__FILE__)."/../M2M/m2m.ini");
$oSettings = new M2M_ServiceConfigurator($oIniFile);
$sOutput .= "<h5>Using URL: " . $oSettings->get_sConnectivityDirectoryUrl() . "</h5>";
$oConnectivityDirectoryClient = new M2M_ConnectivityDirectoryClient($oSettings);
//********************************//
//*** GETCONNECTIVITYDIRECTORY ***//
//********************************//
$oGetConnectivityDirectory= new getConnectivityDirectory();
$oGetConnectivityDirectory->lineIdentifiers= new LineIdentifierCollection();
$oGetConnectivityDirectory->lineIdentifiers->subscriptionNumber = array($sSubscriptionNumber);
//call
$oGetConnectivityDirectoryResponse = $oConnectivityDirectoryClient-
>call_getConnectivityDirectory($oGetConnectivityDirectory);
//Output
$sOutput .=
M2M_Utility::getBeautifiedOutput($oGetConnectivityDirectoryResponse, "getConnectivityDirectory"
);
echo $sOutput;
?>
```

Chapter 5. The M2M SDK

Hereafter is described a global view of the M2M SDK architecture. For a more functional view, please refere to the functional User document.

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

- Common: this library contains reusable classes in the Open Developper Network context.
- M2M: this library contains
  - the `M2mClient` service with access to the M2M API webmethods.
  - the `Proxy` and `Credentials` class, encapsulating authentication data.
  - the Web services, that lets the developper get quickly involved in the M2M API
  - A set of php tools to initialize, to log, to generate outputs of the answers sent by M2M

Figure 5.1. M2M Sdk global architecture diagram

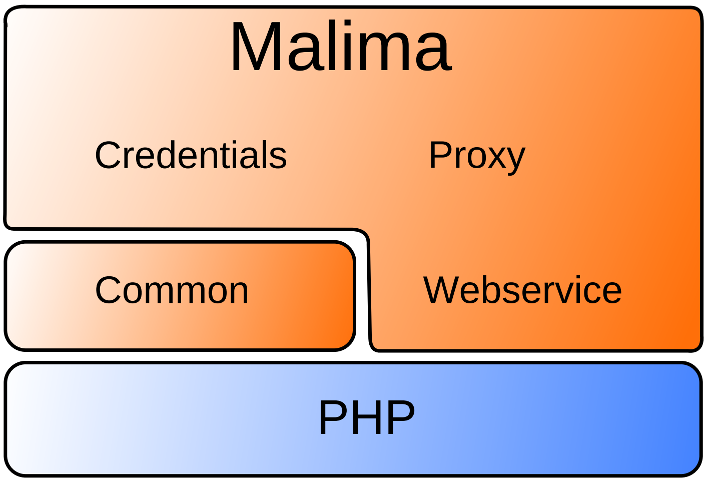
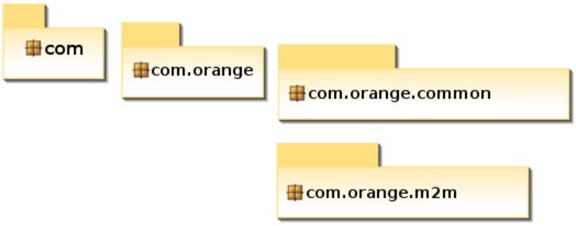


Figure 5.2. M2M Sdk UML model diagram



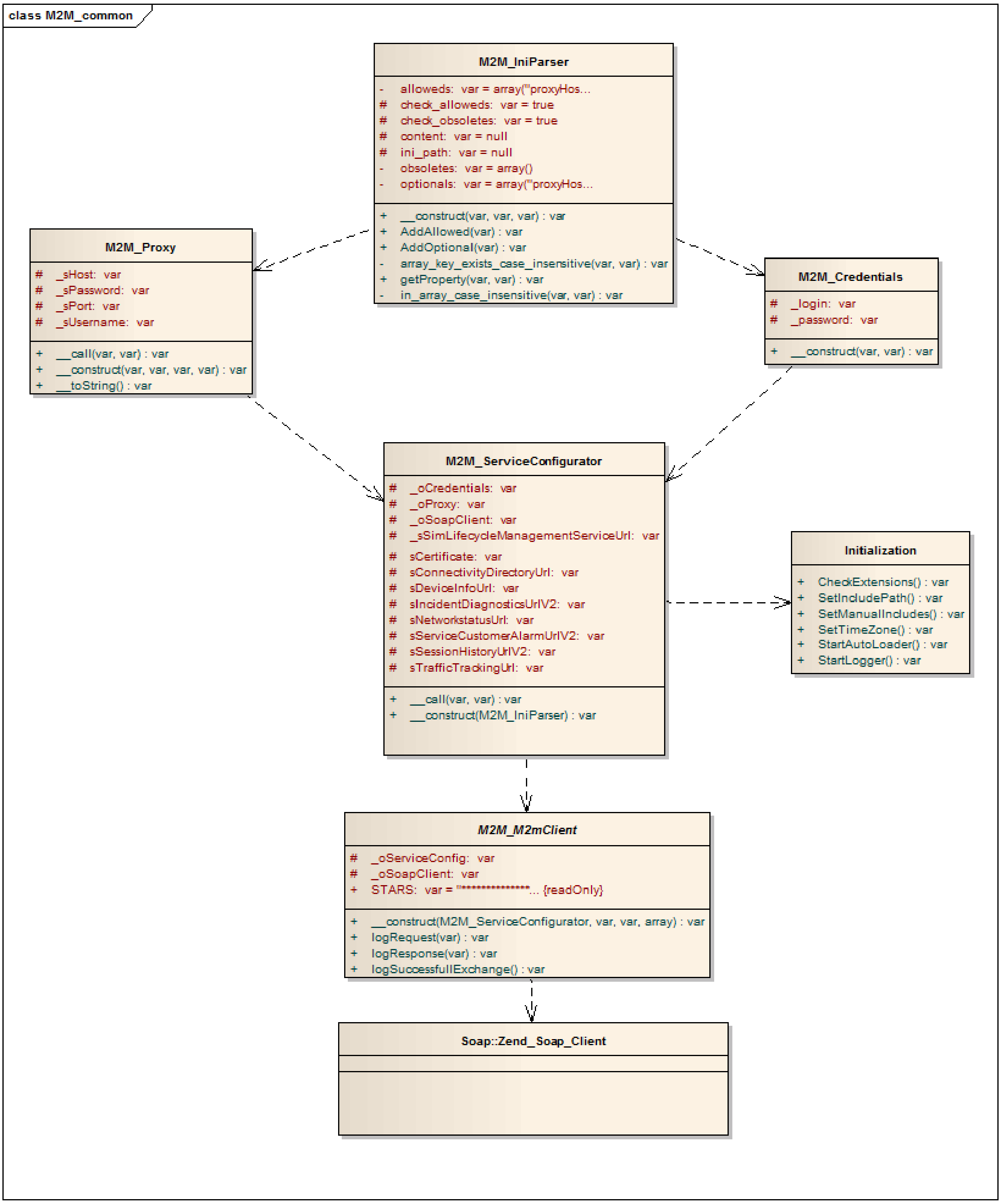
5.1. Common Objects

The M2M folder contains all common required classes. The common classes are IniParser (IniParserException.php),Proxy, Credentials, ServiceConfigurator, Initialization (InitializationException.php), M2mClient (M2mClientException), Zend_Soap_Client(Zend Framework) , FileStream, Utility.

- `IniParser`: this class parses the m2m.ini file to extract the proxy settings, the credentials, the end-points and the log settings.
- `Proxy`: this class encapsulates proxy datas extracted by the IniParser.
- `Credentials`: this class encapsulates credentials datas extracted by the IniParser.

- `ServiceConfigurator`: this class encapsulates all datas contained in Proxy, Credentials objects and the end-points.
- `Initialization`: this class initializes the following methods : `SetIncludePath` `SetTimeZone` `StartAutoLoader` `SetManualIncludes` `StartLogger` `CheckExtensions`
- `M2mClient`: this abstract class is the parent class of each class of service and links ServiceConfigurator, EndPointUrl, WsdlPath and ClassMap. It also manages logs.
- `Zend_Soap_Client`: this Zend Framework class provides us the soap object used.
- `FileStream`: this class extens the Zend Framework class Zend_Log_Writer_Stream which allow us to set our writer.
- `Utility`: this class provides output functionalities.

Figure 5.3. Common classes


class M2M_common
M2M_IniParser
- alloweds: var = array("proxyHos...
# check_alloweds: var = true
# check_obsoletes: var = true
# content: var = null
# ini_path: var = null
- obsoletes: var = array()
- optionals: var = array("proxyHos...
+ __construct(var, var, var) : var
+ AddAllowed(var) : var
+ AddOptional(var) : var
- array_key_exists_case_insensitive(var, var) : var
+ getProperty(var, var) : var
- in_array_case_insensitive(var, var) : var
M2M_Proxy
# _sHost: var
# _sPassword: var
# _sPort: var
# _sUsername: var
+ __call(var, var) : var
+ __construct(var, var, var, var) : var
+ __toString() : var
M2M_Credentials
# _login: var
# _password: var
+ __construct(var, var) : var
M2M_ServiceConfigurator
# _oCredentials: var
# _oProxy: var
# _oSoapClient: var
# _sSimLifecycleManagementServiceUrl: var
# sCertificate: var
# sConnectivityDirectoryUrl: var
# sDeviceInfoUrl: var
# sIncidentDiagnosticsUrlV2: var
# sNetworkstatusUrl: var
# sServiceCustomerAlarmUrlV2: var
# sSessionHistoryUrlV2: var
# sTrafficTrackingUrl: var
+ __call(var, var) : var
+ __construct(M2M_IniParser) : var
Initialization
+ CheckExtensions() : var
+ SetIncludePath() : var
+ SetManualIncludes() : var
+ SetTimeZone() : var
+ StartAutoLoader() : var
+ StartLogger() : var
M2M_M2mClient
# _oServiceConfig: var
# _oSoapClient: var
+ STARS: var = "****************... {readOnly}
+ __construct(M2M_ServiceConfigurator, var, var, array) : var
+ logRequest(var) : var
+ logResponse(var) : var
+ logSuccessfullExchange() : var
Soap::Zend_Soap_Client

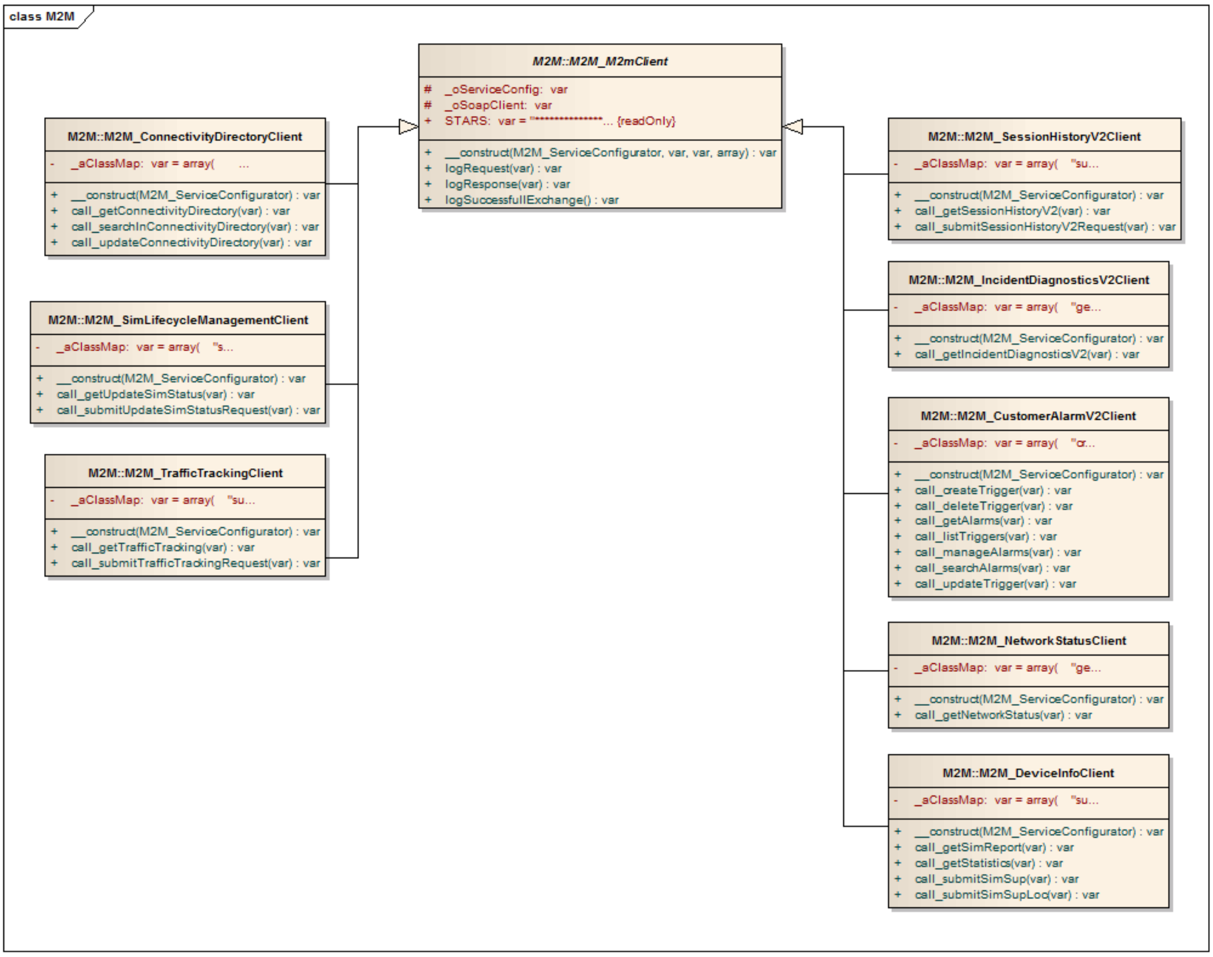
5.2. M2M

The com.orange.M2M package contains several classes.

- `M2M` : 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 :
    - `ConnectivityDirectory` this class provides 3 methods:
        - `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.
    - `UpdateSimStatus` this class provides 2 methods:
        - `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.
    - `TrafficTracking` provides the cumulative total communications established between the network and the machine during the current month. This class provides 2 methods:
        - `submitTrafficTrackingRequest` : Request for a set of SIM cards's traffic in a date range. This method is asynchronous and needs a call to getTrafficTracking method in order to get result.
        - `getTrafficTracking` : Retrieve result of a submitTrafficTrackingRequest according to a ticket number.
    - `SessionHistoryV2` this class provides 2 methods:
        - `submitSessionHistoryV2Request` : Request for a SIM card's set of statistics by sessions in a date range. This method is asynchronous and needs a call to getSessionHistoryV2 method in order to get result.
        - `getSessionHistoryV2` : Retrieve result of a submitSessionHistoryV2Request according to a ticket number.
    - `ServiceCustomerAlarmV2` this class provides 7 methods:
        - `getAlarms` :Get alarms from identifiers.
        - `searchAlarms` :Search alarms launched by triggers on a line or a customer environment for a set of SIM, with or without date criteria.
        - `manageAlarms` : allows the customer to aknowledge his alarms.
        - `createTrigger` : Create a trigger on a set of SIM. Several types of triggers are available, depending on bearer, global threshold, unitary threshold, location, time stamp, lazy machine, silent machine.
        - `updateTrigger` : Update an existing trigger : enabling, disabling, trigger criteria, notifications configuration, ...
        - `deleteTrigger` : Delete an existing trigger.
        - `listTriggers` :Search triggers by identifier, line identifier or customer environment identifier for a set of SIM.
    - `DeviceInfo` this class provides 4 methods:
        - `submitSimSup` : Check if the lines are compatible with the Supervision service and submit a request for supervision.
        - `submitSimSupLoc` : Check if the lines are compatible with the Supervision/Location service and submit a request for supervision/location.
        - `getSimReport` : Get the supervision/location information related to a set of sim cards.
        - `getStatistics` : Get statistics (number of received/sent sms, notifications, sessions in timeout...) for a set of sim cards..
    - `IncidentDiagnosticsV2` this class provides 1 method:
        - `getIncidentDiagnosticsV2` : Get the diagnostic analysis of a SIM card, including the network covering status of the SIM card.

- `NetworkStatus` this class provides 1 method:
  - `getNetworkStatus` : Get the network covering status into an area defined by its latitude and longitude.

Figure 5.4. M2M class diagram.


class M2M
M2M::M2M_M2mClient
# _oServiceConfig: var
# _oSoapClient: var
+ STARS: var = "***************... {readOnly}
+ __construct(M2M_ServiceConfigurator, var, var, array) : var
+ logRequest(var) : var
+ logResponse(var) : var
+ logSuccessfullExchange() : var
M2M::M2M_ConnectivityDirectoryClient
- _aClassMap: var = array(...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getConnectivityDirectory(var) : var
+ call_searchInConnectivityDirectory(var) : var
+ call_updateConnectivityDirectory(var) : var
M2M::M2M_SimLifecycleManagementClient
- _aClassMap: var = array("s...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getUpdateSimStatus(var) : var
+ call_submitUpdateSimStatusRequest(var) : var
M2M::M2M_TrafficTrackingClient
- _aClassMap: var = array("su...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getTrafficTracking(var) : var
+ call_submitTrafficTrackingRequest(var) : var
M2M::M2M_SessionHistoryV2Client
- _aClassMap: var = array("su...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getSessionHistoryV2(var) : var
+ call_submitSessionHistoryV2Request(var) : var
M2M::M2M_IncidentDiagnosticsV2Client
- _aClassMap: var = array("ge...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getIncidentDiagnosticsV2(var) : var
M2M::M2M_CustomerAlarmV2Client
- _aClassMap: var = array("cr...
+ __construct(M2M_ServiceConfigurator) : var
+ call_createTrigger(var) : var
+ call_deleteTrigger(var) : var
+ call_getAlarms(var) : var
+ call_listTriggers(var) : var
+ call_manageAlarms(var) : var
+ call_searchAlarms(var) : var
+ call_updateTrigger(var) : var
M2M::M2M_NetworkStatusClient
- _aClassMap: var = array("ge...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getNetworkStatus(var) : var
M2M::M2M_DeviceInfoClient
- _aClassMap: var = array("su...
+ __construct(M2M_ServiceConfigurator) : var
+ call_getSimReport(var) : var
+ call_getStatistics(var) : var
+ call_submitSimSup(var) : var
+ call_submitSimSupLoc(var) : var

5.3. SDK configuration

The `M2mClient` and its subclasses encapsulate initialization objects named `Credentials` , `Proxy` and `ServiceConfigurator`.

This M2mClient class uses the ServiceConfigurator class which is based on two other classes :

- `Credentials`
- `Proxy`

The ServiceConfigurator class reads a configuration file (`m2m.ini` by default) and set the following parameters:

- the web proxy: you can disable the web proxy or specify your own proxy.
- the credentials: you'll have to set your personnal credentials.
- the service URL.

ServiceConfigurator loads settings from the properties file named `m2m.ini` and is located in the M2M folder.

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

Here is an sample of how to get the configuration datas:

```
     /* Inialization.php manages includes and starts some required functionalities of the Zend
Framework
      * SetIncludePath (Zend)
      * SetTimeZone
      * StartAutoLoader (Zend)
      * SetManualIncludes
      * StartLogger (Zend)
      * CheckExtensions
      */
     $sInitializationPath = dirname ( __FILE__ ) . "/../M2M/Initialization.php";
     if ( ! require_once ($sInitializationPath) ) {
     throw new Exception ( "Cannot find Initialization file at $sInitializationPath" );
     }

     /* M2M_IniParser read the m2m.ini to extract required datas */
     $oIniFile  = new M2M_IniParser(dirname(__FILE__).DIRECTORY_SEPARATOR."/../M2M/m2m.ini"); //
Path to ini file

     /*M2M_ServiceConfigurator contains all configurations datas*/
     $oSettings = new M2M_ServiceConfigurator($oIniFile);
```

Here is an example on how to get the Connectivity Directory Url from the m2m.ini file.You can acces to a dedicated data this way :

```
   $oSettings->get_sConnectivityDirectoryUrl();
```

You can find this example in the ConnectivityDirectoryExample.php code snippet provided with the sdk located in the sample folder.

5.4. Logging

The SDK uses the Zend Framework 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 (by default, logging is disabled).

Zend_Log is a tool to help the programmer output log statements to a variety of output targets. .

Typically the Zend_Log activation is specified using a properties file.

The following content shows you the properties file `m2m.ini` that is shipped with the SDK where you can enabled the Zend logger.

```
     ;-------------------------------------------------------------
     ; (3) Misc
     ;-------------------------------------------------------------
     ; set it to "no" to log.
     DisableLogger = yes
```

Here's a sample of Initialization.php where the zend logger is loaded :

```
$log = new Zend_Log ( );

// Formatting
$defaultFormat = Zend_Log_Formatter_Simple::DEFAULT_FORMAT;
$format = $defaultFormat . '%client_ip% %user_agent%' . PHP_EOL;

$path = dirname ( __FILE__ ) . "/../logs/events.log";
if ( file_exists ( dirname ( $path ) ) ) {
$writer = new M2M_FileStream ( $path );
$writer->setFormatter ( new Zend_Log_Formatter_Simple ( $format ) );
$log->addWriter ( $writer );
}

// Adding parameters to log: IP and browser
if ( array_key_exists ( 'HTTP_USER_AGENT', $_SERVER ) ) {
$log->setEventitem ( 'user_agent', $_SERVER ['HTTP_USER_AGENT'] );
}
if ( array_key_exists ( 'REMOTE_ADDR', $_SERVER ) ) {
$log->setEventitem ( 'client_ip', $_SERVER ['REMOTE_ADDR'] );
}

Zend_Registry::set ( 'log', $log );
```

You can create your personnal log file, and set path to it as shown above. By default, you can find the log file `event.log` in the "logs" folder of the sdk.

This log configuration file defines a writer which write into the file `event.log` but Zend_log provides other way to log :

- directly in the output of your browser
- in a database
- firebug [https://addons.mozilla.org/fr/firefox/addon/1843] console

You can find more information about Zend_log here [http://framework.zend.com/manual/fr/zend.log.writers.html]

You'll find hereafter an example of a Zend_log output:

```
***************************SOAP request HTTP header: POST /MLM/SessionHistory-1 HTTP/1.1
Host: 8X.1X.21X.20X
Connection: Keep-Alive
User-Agent: PHP-SOAP/5.2.11
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-Length: 662
Authorization: Basic NjZNM1YzMkQ4N1ZVOVBLOVXXXXXXXXXXelY
127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
2010-03-22T10:31:58+01:00 DEBUG (7):

***************************SOAP request HTTP body:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="http://common.types.malima.francetelecom.com"
```

```
        xmlns:ns2="http://sessionHistory.types.malima.francetelecom.com"
        xmlns:ns3="http://webservice.malima.francetelecom.com/v1">
        <SOAP-ENV:Body>
          <ns3:submitSessionHistoryRequest>
            <ns2:lineIdentifier>
              <ns1:subscriptionNumber>23697253</ns1:subscriptionNumber>
            </ns2:lineIdentifier>
            <ns2:periodStartDateTime>2010-01-11T08:02:09+01:00</ns2:periodStartDateTime>
            <ns2:periodEndDateTime>2010-02-11T08:02:09+01:00</ns2:periodEndDateTime>
          </ns3:submitSessionHistoryRequest>
        </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>

       --- Using style  and using use
      127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
      2010-03-22T10:31:58+01:00 DEBUG (7): Success
      127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
      2010-03-22T10:31:58+01:00 DEBUG (7):

      ***************************SOAP response HTTP header: HTTP/1.1 200 OK

      X-Backside-Transport: OK OK
      Connection: Keep-Alive
      Transfer-Encoding: chunked
      Server: Apache-Coyote/1.1
      Content-Type: text/xml; charset=UTF-8
      Date: Mon, 22 Mar 2010 09:29:55 GMT
      X-Client-IP: 1X.6X.7X.230

      127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
      2010-03-22T10:31:58+01:00 DEBUG (7):

      ***************************SOAP response HTTP body:
      <?xml version="1.0" encoding="UTF-8"?>
      <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Header>
          <t:trackingHeader
            xmlns:t="http://www.francetelecom.com/iosw/v1"
            xmlns:date="http://exslt.org/dates-and-times">
            <t:requestId>c6e4e2c6-a470-4b64-b4db-6d238408f7a8</t:requestId>
            <t:timestamp>2010-03-22T10:29:55+01:00</t:timestamp>
          </t:trackingHeader>
        </soap:Header>
        <soap:Body>
          <ns2:submitSessionHistoryRequestResponse
            xmlns:ns2="http://webservice.malima.francetelecom.com/v1"
            xmlns:ns3="http://common.types.malima.francetelecom.com"
            xmlns:ns4="http://sessionHistory.types.malima.francetelecom.com"
            xmlns:ns5="http://exception.malima.francetelecom.com"
            xmlns:ns6="http://networkstatus.types.malima.francetelecom.com">
            <ns4:ticketNumber>400</ns4:ticketNumber>
          </ns2:submitSessionHistoryRequestResponse>
        </soap:Body></soap:Envelope>
      127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
      2010-03-22T10:31:58+01:00 DEBUG (7): Success
      127.0.0.1 Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6
      2010-03-22T10:31:58+01:00 DEBUG (7):

(...)
```

Chapter 6. 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.

6.1. 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.

Before calling any web service, you need to initialize the parameters (endpoints, credentials, proxy), using the `M2M_IniParser` and a `M2M_ServiceConfigurator` classes.

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

1. Create a `M2M_ConnectivityDirectoryClient` object.
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 `call_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 `M2M_ConnectivityDirectoryClient` object.
2. Create an `updateConnectivityDirectory` object for a set of SIM cards.
3. Create a `dataBySubscriptionNumber` or `dataBySimSerialNumber` or `dataByDeviceImei` or `dataByDeviceSerialNumber` or `dataByMsisdnVoice` or `dataByMachineSerialNumber` or `dataByDeviceUniqueId` or `dataByMsisdnData` or `dataByMsisdnFax` object (or array of objects), depending on the parameter you want to use as identifier.
4. In the end, call the `call_updateConnectivityDirectory` API method using the `updateConnectivityDirectory` object.

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

1. Create a `M2M_SimLifecycleManagementClient` object.
2. Create a `submitUpdateSimStatusRequest` object and a `LineIdentifierCollection` object for a set of SIM cards.
3. Set the requested status for the set of SIM cards (ACTIVATED_FOR_TEST, ACTIVATED, SLEEPING, SUSPENDED).
4. Set the test mode for the area covered (INTERNATIONAL, EUROPE, METROPOLE, AU_COMPTEUR).
5. Call the asynchronous `call_submitUpdateSimStatusRequest` API method.
6. Get the ticketNumber returned.
7. Call the `call_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 session history of a given SIM cards ?

1. Create a `M2M_SessionHistoryClient` object.
2. Create a `submitSessionHistoryRequest` object with the attached `LineIdentifier` for the SIM card.

3. Create a `start date` and an `end date` for a given period.
4. Call the asynchronous `call_submitSessionHistoryRequest` API method.
5. Get the `TicketNumber` returned.
6. Call the `call_getSessionHistory` 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 manage alarms and triggers for a set of SIM cards (V2) ?

Please note that there are two versions of `CustomerAlarm` API method, one corresponding to the V1 version, and the other to the V2 version.

This SDK can manage both but please use V2 version because the V1 version will soon be obsolete.

1. Create a `M2M_CustomerAlarmV2Client` object.

   [Define and Manage Trigger]
2. Create a trigger, calling the `call_createTrigger` API method, by giving a set of information for the trigger : a comment, a level, a notification, a groupid, a scope (customer environment or line identifier), a criteria (timestamp, location, global threshold, unitary threshold, bearer, update imei).
3. Get the list of all existing triggers, calling the `call_listTriggers` API method, by giving a list of trigger identifiers, of line identifiers or a customer environment number. You will know everything about each trigger : comment, level, notification, groupid, scope (customer environment or line identifier), criteria (timestamp, location, global threshold, unitary threshold, bearer, update imei).
4. If necessary, update a trigger, calling the `call_updateTrigger` API method, in order to enable the trigger, or update the trigger criteria. You can add new allowed bearers (SMS, MMS, WIFI, VOICE, ...) in a bearer criteria for instance.

   [Configure Notifications (Optional)]
5. Get prepared to receive a URL call when a alarm will be raised.

   [Get Alarms raised]
6. Search alarms launched by triggers on a line, or a customer environment, with or without date criteria, calling the `call_searchAlarms` API method.
7. After receiving a URL call on the notification URL, get alarms launched by triggers by giving alarm identifiers, calling the `call_getAlarms` API method.
8. The end-user can acknowledge his alarms, calling the `call_manageAlarms` API method.

   [Delete a Trigger]
9. Delete a trigger, calling the `call_deleteTrigger` API method, by giving a trigger identifier, in order to delete the trigger.

What are the basic steps to get Device Information on a set of SIM cards ?

1. Do a `getConnectivityDirectory` on the sim cards in order to check if they are compatible with the service (don't forget to set the *showDeviceInfo* flag to true in your request)
2. Call the `submitSimSupLoc` method to run a monitoring AND location request on the sim cards. The response will contain a ticket number. if you do not want to locate your sim cards, you can use the `submitSimSup` method instead.
3. Call the `getSimReport` method, with the ticket number in parameter, in order to get the results for your monitoring/location request.
4. Call the `getStatistics` method if you wish to get information on how many SMS were sent during your monitoring/location requests.

What are the basic steps to get your network status ?

1. Create a `M2M_NetworkStatusClient` object.
2. Create a `Coordinates` object corresponding to GPS coordinates of the area you where want the network status.
3. In the end, call the `call_getNetworkStatus` API method. You will get information about the network coverage in the location, for each kind of network technology (GPRS, UMTS...).

What are the basic steps to get incident disgnostics (V2) of a SIM card ?

1. Create a `M2M_IncidentDiagnosticsV2Client` object.
2. Create a `getIncidentDiagnostics` object and the attached `LineIdentifier` for the SIM card.
3. At last, call the `call_getIncidentDiagnosticsV2` API method. You will get information related to the location of the SIM card, the network coverage in the location, incidents that might be on-going in the area.

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

1. Create a `M2M_TrafficTrackingClient` object.
2. Create a `submitTrafficTrackingRequest` object and 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 `call_submitTrafficTrackingRequest` API method.
5. Get the `TicketNumber` returned.
6. Call the `call_getTrafficTracking` API method until the global response status of the method is either TERMINATED or ERROR, meaning the asynchronous generation has ended.

beginner

6.2. Self Management

6.2.1. Connectivity Directory

6.2.1.1. `GetConnectivityDirectory` method

6.2.1.1.1. Description

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

6.2.1.1.2. Input parameter

Table 6.1. getConnectivityDirectory: Input parameters.

Type	Description	Cardinality
`getConnectivityDirectory`	This object contains a LineIdentifierCollection object : a collection of Line Identifiers or An array of String containing the imsi identifiers	1..1

6.2.1.1.3. Output parameter

Table 6.2. getConnectivity Directory: output parameters.

Type	Description	Cardinality
`getConnectivity DirectoryResponse`	This object containts an array of ConnectivityDirectory and/or a LineIdentifierCollection (collection of unknown line identifiers) and/or an array of unknown imsi identifiers.	1..1

6.2.1.1.4. Exceptions

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

6.2.1.2. `SearchIn Connectivity Directory` method

6.2.1.2.1. Description

The `SearchIn Connectivity Directory` operation enable to search SIM cards according to search criterions.

6.2.1.2.2. Input parameter

Table 6.3. searchIn Connectivity Directory: Input parameters.

Type	Description	Cardinality
`searchInConnectivity Directory`	This object contains ConnectivityDirectory SearchCriterion, rangeSize (int), rangeStart (int), SortCriterion, showStatSimPerSimStatus (boolean) objects	1..1

beginner

6.2.1.2.3. Output parameter

Table 6.4. searchInConnectivity Directory: output parameters.

Type	Description	Cardinality
`searchIn Connectivity DirectoryResponse`	The response made of an array of ConnectivityDirectory objects, SimStatusDistribution object (sim number per sim status), the global number of results, and a boolean telling whether the number of results has exceeded or not	1..1

6.2.1.2.4. Exceptions

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

6.2.1.3. `Update Connectivity Directory` method

6.2.1.3.1. Description

The `Update Connectivity Directory` operation updates a set of connectivity directory data, managed by different kinds of identifiers.

6.2.1.3.2. Input parameter

Table 6.5. update Connectivity Directory: Input parameters.

Type	Description	Cardinality
`updateConnectivity 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

6.2.1.3.3. Output parameter

Table 6.6. update Connectivity Directory: output parameters.

Type	Description	Cardinality
`updateConnectivity DirectoryResponse`	An array of connectivity directories updated successfully (UpdatesCompletedList), and an array of connectivity directories for which the update failed (UpdatesFailedList)	1..1

6.2.1.3.4. Exceptions

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

6.2.2. Sim Lifecycle Management (a.k.a UpdateSimStatus)

6.2.2.1. `SubmitUpdateSimStatus` method

6.2.2.1.1. Description

Submit an update of SIM card status for a set of SIM cards.

beginner

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

6.2.2.1.2. Input parameters

Table 6.7. submitUpdateSimStatus: Input parameters.

Type	Description	Cardinality
`submitUpdate SimStatusRequest`	This object contains a LineIdentifierCollection, requestedStatus (ACTIVATED_FOR_TEST, ACTIVATED, SLEEPING, SUSPENDED), testMode (INTERNATIONAL, EUROPE, METROPOLE or AU_COMPTEUR)	1..1

6.2.2.1.3. Output parameter

Table 6.8. submitUpdateSimStatus: output parameters.

Type	Description	Cardinality
`submitUpdate SimStatusRequest Response`	A ticket number useable in the getUpdateSimStatus request, and a collection of unknown requested LineIdentifiers if any.	1..1

6.2.2.1.4. Exceptions

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

6.2.2.2. `GetUpdateSimStatus` method

6.2.2.2.1. Description

Get the result of a `SubmitUpdateSimStatus` method call.

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.

6.2.2.2.2. Input parameters

Table 6.9. getUpdateSimStatus: Input parameters.

Type	Description	Cardinality
`getUpdate SimStatusResult`	This object contains a ticket number returned by the submitUpdateSimStatus method call.	1..1

beginner

6.2.2.2.3. Output parameter

Table 6.10. getUpdateSimStatus: output parameters.

Type	Description	Cardinality
`getUpdate SimStatus ResultResponse`	A global ticket status telling wether the request is being processed, in progress or terminated ; an array of StatusUpdate (one for each requested LineIdentifier); a reminder of SlmSimStatus and TestMode values ; an array of unknown line identifier if any.	1..1

6.2.2.2.4. Exceptions

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

6.2.3. Traffic Tracking

6.2.3. Traffic Tracking

provide the cumulative total communications established between the network and the machine during the current month. It returns these cumulative totals for the SIM cards requested in the request. The cumulative totals are provided by bearer and by origin (roaming or local or local+roaming) for communications received since the beginning of the month only.

6.2.3.1. `SubmitTrafficTracking` method

6.2.3.1.1. Description

Input parameters :

- A list of SIM cards (Type and value)
- Date and end of consultation period: these are the start and end dates for receipt of communications in the system
  - If no date is provided then the service uses the first day of the current month as the period start date; the period end date is then the input date,
  - If only the start date is given, the end date used is the current date
  - If only the end date is given, the start date used is the first day of the current month
  - If the start date entered is prior to the first day of the current month then the service limits the start of the search to the first day of the current month
  - If the end date is after the input date then the service limits the end of the search to the input date

6.2.3.1.2. Input parameter

Table 6.11. submitTrafficTracking: Input parameters.

Type	Description	Cardinality
`submitTraffic TrackingRequest`	This object contains a collection of Line Identifiers, a start date and end date	1..1

6.2.3.1.3. Output parameter

Table 6.12. submitTrafficTracking: output parameters.

Type	Description	Cardinality
`submitTraffic TrackingRequestResponse`	This object contains a ticket number useable in the getTrafficTracking request, and a collection of unknown requested LineIdentifiers if any.	1..1

6.2.3.1.4. Exceptions

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

6.2.3.2. `GetTrafficTracking` method

6.2.3.2.1. Description

Get the result of a `SubmitTrafficTracking` method call.

The method `GetTrafficTracking` has to be used after a `SubmitTrafficTracking` 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.

6.2.3.2.2. Input parameter

Table 6.13. getTrafficTracking: Input parameters.

Type	Description	Cardinality
`getTrafficTracking`	contains a ticket number returned by the submitTrafficTracking method, and a showAllOriginSum boolean (returns local + roaming traffic id set to true). call.	1..1

6.2.3.2.3. Output parameter

Table 6.14. getTrafficTracking: output parameters.

Type	Description	Cardinality
`getTraffic TrackingResponse`	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 Traffic, 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

6.2.3.3. Exceptions

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

beginner

6.3. Self diagnosis

6.3.1. Alarms V2

6.3.1.1. `CreateTrigger` method

6.3.1.1.1. Description

The `CreateTrigger` operation enables to create a new trigger.

6.3.1.1.2. Input parameters

Table 6.15. CreateTrigger: Input parameters.

Type	Description	Cardinality
`createTrigger`	A set of information about the new trigger : a comment, a level, a notification, a groupid, a scope (customer environment or line identifier), a criteria (timestamp, location, global threshold, unitary threshold, bearer, update imei, lazy machine, silent machine).	1..1

6.3.1.1.3. Output parameter

Table 6.16. CreateTrigger: output parameter.

Type	Description	Cardinality
`CreateTriggerResponse`	contains the identifier for the trigger newly created.	1..1

6.3.1.1.4. Exceptions

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

6.3.1.2. `DeleteTrigger` method

6.3.1.2.1. Description

The `DeleteTrigger` operation enable to delete an existing trigger.

6.3.1.2.2. Input parameters

Table 6.17. DeleteTrigger: Input parameters.

Type	Description	Cardinality
`deleteTrigger`	The identifier of the trigger to delete.	1..1

6.3.1.2.3. Output parameter

Table 6.18. DeleteTrigger: output parameter.

Type	Description	Cardinality
`deleteTriggerResponse`	The identifier of the deleted trigger.	1..1

6.3.1.2.4. Exceptions

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

6.3.1.3. `UpdateTrigger` method

6.3.1.3.1. Description

The `UpdateTrigger` operation enables to update an existing trigger.

6.3.1.3.2. Input parameters

Table 6.19. UpdateTrigger: Input parameters.

Type	Description	Cardinality
`updateTrigger`	A set of information identifying the trigger to update, and the data to update : status, comment, level, notification, groupid, scope (customer environment or line identifier), criteria.	1..1

6.3.1.3.3. Output parameter

Table 6.20. UpdateTrigger: output parameter.

Type	Description	Cardinality
`UpdateTriggerResponse`	The identifier of the updated trigger.	1..1

6.3.1.3.4. Exceptions

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

6.3.1.4. `ListTriggers` method

6.3.1.4.1. Description

The `ListTriggers` operation enables to list triggers.

6.3.1.4.2. Input parameters

Table 6.21. ListTriggers: Input parameters.

Type	Description	Cardinality
`listTriggers`	The identifier of the trigger to list.	1..1

6.3.1.4.3. Output parameter

Table 6.22. ListTriggers: output parameter.

Type	Description	Cardinality
`listTriggersResponse`	contains a list of triggers with information about each trigger : status, comment, level, notification, groupid, scope (customer environment or line identifier), criteria.	1..1

6.3.1.4.4. Exceptions

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

6.3.1.5. SearchAlarms method

6.3.1.5.1. Description

The SearchAlarms operation enable to search alarms.

6.3.1.5.2. Input parameters

Table 6.23. SearchAlarms: Input parameters.

Type	Description	Cardinality
searchAlarms	Search criteria : date, range, target (line identifiers or customer environment number) or trigger identifiers list.	1..1

6.3.1.5.3. Output parameter

Table 6.24. SearchAlarms: output parameter.

Type	Description	Cardinality
searchAlarms Response	A list of alarms	1..1

6.3.1.5.4. Exceptions

This web method may throw TechnicalException or FunctionalException.

6.3.1.6. GetAlarms method

6.3.1.6.1. Description

The GetAlarms operation enable to get details about alarms.

6.3.1.6.2. Input parameters

Table 6.25. GetAlarms: Input parameters.

Type	Description	Cardinality
getAlarms	contains a set of alarm identifiers.	1..1

6.3.1.6.3. Output parameter

Table 6.26. GetAlarms: output parameter.

Type	Description	Cardinality
getAlarmsResponse	The detail of each alarm wanted : alarm identifier, creation date, level, customer environment number, subscription number, notification date, notification result code, notified Url, notification result message, number alarm exceeded, triggered alarm, notification required, trigger identifier, trigger comment, trigger groupId, trigger type, trigger criteria, ...	1..1

6.3.1.6.4. Exceptions

This web method may throw TechnicalException or FunctionalException.

beginner

6.3.1.7. ManageAlarms method

6.3.1.7.1. Description

The ManageAlarms operation enables the end-user to acknowledge his alarms.

6.3.1.7.2. Input parameters

Table 6.27. ManageAlarms: Input parameters.

Type	Description	Cardinality
manageAlarms	contains the list of alarms to be acknowledged.	1..1

6.3.1.7.3. Output parameter

Table 6.28. ManageAlarms: output parameter.

Type	Description	Cardinality
manageAlarmsResponse	A list of alarms and/or unknown alarms and/or alarms with unknwon status	1..1

6.3.1.7.4. Exceptions

This web method may throw TechnicalException or FunctionalException.

6.3.1.8. Sample : notifications

This sample uses the m2m.ini file, don't forget to fill in your information.

It explains how getting details about a raised alarm, when receiving a URL call on the notification URL.

6.3.2. Device supervision/location

6.3.2.1. submitSimSup method

6.3.2.1.1. Description

Check if the lines are compatible with the Supervision service and submit a request for supervision/location.

6.3.2.1.2. Input Parameters

Table 6.29. Input Parameters

Output object name	Output object type	Description	Card.
lineIdentifierCollection	LineIdentifierCollection	a collection of subscription identifiers.	1..n

6.3.2.1.3. Output Parameter

Table 6.30. Output Parameters

Output object name	Output object type	Description	Card.
response	SubmitSimSupResponse	Ticket Number and/or a list of unknown line identifiers.	1..1

6.3.2.2. submitSimSupLoc method

6.3.2.2.1. Description

Check if the lines are compatible with the Supervision/Location service and submit a request for supervision/ location.

6.3.2.2.2. `Input Parameters`

Table 6.31. Input Parameters

Output object name	Output object type	Description	Card.
`lineIdentifierCollection`	`LineIdentifierCollection`	a collection of subscription identifiers.	1..n

6.3.2.2.3. `Output Parameter`

Table 6.32. Output Parameters

Output object name	Output object type	Description	Card.
`response`	`SubmitSimSupLocResponse`	Ticket Number and/or a list of unknown line identifiers.	1..1

6.3.2.3. getSimReport method

6.3.2.3.1. Description

Check if the lines are compatible with the Supervision/Location service and submit a request for supervision/ location.

6.3.2.3.2. `Input Parameters`

Table 6.33. Input Parameters

Output object name	Output object type	Description	Card.
`ticketNumber`	`int`	The ticket number retrieved in the submitSup/submitSupLoc response.	1..1

6.3.2.3.3. `Output Parameter`

Table 6.34. Output Parameters

Output object name	Output object type	Description	Card.
`response`	`GetSimReportResponse`	The ticket global status, a list of unknown line identifiers (if any) and the supervision (connectivity, signal quality, identity)/location detailed information.	1..1

beginner

6.3.2.4. getStatistics method

6.3.2.4.1. Description

Returns statistics (number of received/sent sms, notifications, sessions in timeout...) for a set of sim cards.

6.3.2.4.2. `Input Parameters`

Table 6.35. Input Parameters

Output object name	Output object type	Description	Card.
`period`	`Period[]`	Each period contains 2 dates : search period start date and period end date.	1..n
`lineIdentifierCollection`	`LineIdentifierCollection`	A collection of line identifiers.	1..n

6.3.2.4.3. `Output Parameter`

Table 6.36. Output Parameters

Output object name	Output object type	Description	Card.
`response`	`GetStatisticsResponse`	contains, for each sim card, the detailed statistics, a list of unknown line identifiers, and a list of validation errors (wrong period, line not compatible with the supervision/location service...).	1..1

6.3.3. Incident Diagnostics

6.3.3.1. Description

Get an Incident diagnostics information concerning a Line Identifier.

6.3.3.2. Input parameter

Table 6.37. getIncidentDiagnosticsV2: Input parameters.

Type	Description	Cardinality
`getIncidentDiagnostics`	contains a Line Identifier	1..1

6.3.3.3. Output parameter

Table 6.38. getIncidentDiagnosticsV2: output parameters.

Type	Description	Cardinality
`getIncident DiagnosticsResponse`	An IncidentDiagnosticsV2 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

beginner

6.3.3.4. Exceptions

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

6.3.4. Network Status

6.3.4.1. Description

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

6.3.4.2. Input parameter

Table 6.39. getNetworkStatus: Input parameters.

Type	Description	Cardinality
`getNetworkStatus`	contains GPS coordinates in WGS84 / DMS0 format. Examples : N48d48m41s0.0 - E2d19m39s0.0 N45d0m14s0.0 - W1d11m49s0.0	1..1

6.3.4.3. Output parameter

Table 6.40. getNetworkStatus: output parameters.

Type	Description	Cardinality
`getNetworkStatus Response`	A Network Stat Status telling wether 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 occuring, with a brief description.	1..1

6.3.4.4. Exceptions

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

6.3.4.4.1. Legend

Table 6.41. Legend for NetworkStatus errors

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

6.3.5. Session History V2

These methods allow users to get the details of communications exchanged between the network and the SIM card.

beginner

6.3.5.1. Description

Input parameters : a SIM card identifier (Type and value)

- The search period begin and end dates.
  - Both dates are required
  - The two dates and times must be different
  - The end date must be later than the start date
  - The start date must not be earlier than 180 days before the current date
  - The period may not exceed 31 days (time between start date and end date)
- If the identifier or one of the two dates is missing then an error message is sent.
  - invalidPeriodErrorFault if one of the two dates is missing
  - malimaErrorFault = MLM_UNKNOWN_USER is the SIM card identifier is missing
- If the dates do not adhere to the specified rules, an error message is returned:
  - invalidPeriodErrorFault: type of error if the period is invalid
  - The Values may be:
    - The end date is before the start date
    - the start date is earlier than 180 days
    - The delay between the start date and the end date exceeds 31 days

The getSessionHistoryResponse is paginated: in the getSessionHistory response, if the field **NumberOfRemainingResults** is higher than 0, you must perform additional getSessionHistory request(s) in order to get remaining results. Take the current ticket number as input parameter.

Example : a getSessionResponse returns :

- Period start date time : 01/01/2014 00:00:00
- Period end date time : 18/01/2014 00:00:00
- Number of remaining results : 9
- Total number of results : 59

It the emission date for the last CDR returned in the request is Jan. 17th 2014 13:49:54, then, in order to retrieve the last 9 sessions, you need to create another getSessionHistory request with the current ticket id as parameter.

6.3.5.2. `SubmitSessionHistoryV2` method

6.3.5.2.1. Description

The method `SubmitSessionHistoryV2` is asynchronous and has to be used prior the `getSessionHistoryV2` method cal which gets back the submit result.

6.3.5.2.2. Input parameter

Table 6.42. submitSessionHistoryV2: Input parameters.

Type	Description	Cardinality
`submitSessionHistoryRequest`	contains a unique Line Identifier, a start date and an end date.	1..1

6.3.5.2.3. Output parameter

Table 6.43. submitSessionHistoryV2: output parameters.

Type	Description	Cardinality
`submitSessionHistoryResponse`	contains a ticket number useable in the getSessionHistoryV2 request.	1..1

6.3.5.2.4. Exceptions

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

6.3.5.3. `GetSessionHistoryV2` method

6.3.5.3.1. Description

Get the result of a `SubmitSessionHistoryV2` method call.

The method `GetSessionHistoryV2` has to be used after a `SubmitSessionHistoryV2` 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.

6.3.5.3.2. Input parameter

Table 6.44. getSessionHistoryV2: Input parameters.

Type	Description	Cardinality
`getSessionHistory`	contains the ticket number returned by the submit method call.	1..1

6.3.5.3.3. Output parameter

Table 6.45. getSessionHistory: output parameters.

Type	Description	Cardinality
`getSessionHistoryResponse`	contains a global ticket status telling wether the request is being processed, in progress or terminated ; a Line data (refering an array of CustomerData, a Subscription, a SIM card, a Device, a machine identifier) ; and an array of Session.	1..1

6.3.5.4. Exceptions

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

Appendix A. Error codes

A.1. Malima error codes

Table A.1. Malima error codes

Code	Detail
`malimaErrorFault`	• CANCEL_BY_ADMIN :“The ticket was canceled by an administrator”. • CANNOT_CONNECT_TO_CDR_DATABASE : “Cannot connect to the cdr database.” • CANNOT_CONNECT_TO_DATABASE : “Cannot connect to the core database.” • INVALID_RANGE_SIZE_PARAMETER : “ The range size <provided range size> cannot be less than <minimum allowed range size> ” • INVALID_RANGE_START_PARAMETER “ The range start <provided range start> cannot be less than <minimum allowed range start> ” • MALIMA_INTERNAL_ERROR “A malima internal error has occurred. ” • MLM_DEFAULT_ERROR_CODE “A Malima plateform error has occured. ” • MLM_DEFAULT_NEXT_STATUS_FORBIDDEN “Forbidden default next status is present. ” • MLM_INVALID_OPTION_PARAMETER “ Invalid parameter <code of the parameter> for option <code of the option> ” • MLM_MANDATORY_VALUE_FOR_OPTION_PARAMETER “ An empty or null value is not acceptable unlike <NULL> which allows remove the parameter value ” • MLM_MISSING_OPTION_PARAMETER “ Missing mandatory parameter <code of the parameter> for option <code of the option> ” • MLM_MISSING_REQUIRED_DEFAULT_NEXT_STATUS “Required default next status is missing. ” • MLM_NOT_AUTHORIZED_TRANSITION “ Transition between status <source status> and status <target status> is not authorized ” • MLM_QUOTA_EXCEEDED “ Quota exceeded: it remains just <remaining authorized hit count> / <allowed hit count> hit for this web service. ”
`malimaErrorFault`	• MLM_QUOTA_EXCEEDED “ Remaining quota : <N> request(s).” • MLM_QUOTA_EXCEEDED “ you have <remaining authorized hit count> hits left for this service until the end of the month ” • MLM_RES_ALREADY_RETR “ The result has already been retrieved : <the full retrieval date> ” • MLM_SOAP_HEADER_ERROR : “specific message” • MLM_SOAP_HEADER_ERROR “The user <node id> is unknown to Malima or Portal header in SOAP message is missing or Some input parameters in portal header are not valid : <missing field> or Portal header in SOAP message is missing or Some input parameters in portal header are not valid : <missing field> ” • MLM_SOAP_HEADER_ERROR “ Wrong parameter from IOSW-IAS call : <header fields in error> ” • MLM_TICKET_CANCELLED_BY_ADMIN “The ticket was cancelled by an administrator MLM_DEFAULT_ERROR_CODE A Malima plateform error has occured. ” • MLM_TICKET_CREATION “ticket creation failed ” • MLM_TICKET_CREATION_ERROR “The ticket creation has failed” • MLM_TOO_MANY_NODE_IDENTIFIERS “The number of node identifiers is too high. (<N>maximum authorized)”

Code	Detail
	• MLM_UNKNOWN_BEARER " The bearer <b> is unknown in the Malima configuration. " • MLM_UNKNOWN_REQUESTED_SIM_STATUS " Requested SIM status <SIM status> unknown " • MLM_UNKNOWN_SORT_ATTRIBUTE " The sort attribute <sort attribute> is unknown in Malima. " • MLM_UNKNOWN_USER "The user <node id>is unknown to Malima" • MLM_UNKNOWN_USER "user <user name>is unknown"
`malimaErrorFault (customized fields)`	• MLM_CUSTOM_CD_FIELDS_ALREADY_EXIST "The customized field <label> already exists." • MLM_DUPLICATE_CD_CUSTOM_FIELD_LABEL "The following customized fields are in double: <list_duplicate_cd_custom_field>" • MLM_DUPLICATE_CD_CUSTOM_POSSIBLE_VALUE " The following customized possible values are in double: <list_duplicate_cd_custom_possible_values> " • MLM_TOO_MANY_CD_CUSTOM_FIELD : "The number of customized fields is limited to <CD_MAX_CUSTOMIZED_FIELDS>" • MLM_TOO_MANY_CD_CUSTOM_POSSIBLE_VALUE "The number of possible values is limited to <CD_MAX_CUSTOMIZED_POSSIBLE_VALUE>" • MLM_BAD_FORMAT_CD_CUSTOM_FIELD "Bad format of custom field: <bad_custom_field>. Expected format is: <format>." • MLM_BAD_FORMAT_CD_CUSTOM_POSSIBLE_VALUE "Bad format of possible values: <list_bad_possible_values>. Expected format is: <format>. " • MLM_UNKNOWN_CD_CUSTOM_FIELD "Unknown customized fields: <unknown_field_list>" • MLM_DUPLICATE_CD_CUSTOM_FIELD_LABEL "The following customized fields are in double: <list_duplicate_cd_custom_field>. "
`malimaErrorFault (triggers/ alarms)`	• MLM_BAD_M2MSUB_LIST"The input M2M subscription status list is empty or contains incorrect status that could not be triggered." • MLM_BAD_OPTION_LIST "The input subscription option list is empty or contains option that could not be triggered." • MLM_DUPLICATE_CODE_OPTION " The trigger could not be created, the following option code are in double : <option code>. " • MLM_BAD_OPTION_TYPE_EVENT : "The input option change type could not be triggered.;" • MLM_TOO_MANY_CD_CUSTOM_POSSIBLE_VALUE "The number of possible values is limited to <CD_MAX_CUSTOMIZED_POSSIBLE_VALUE>" • MLM_DUPLICATE_OPTION_TYPE_EVENT "The trigger could not be created, the following option change type event are in double : <option code>."

A.2. Error Codes synthesis

A.2.1. Connectivity Directory - Error codes

Table A.2. Error codes and Services

Error	getCD	searchInCD	updateCD
`MalimaError`	X	X	X
`TooMany LineIdentifiersError`	X		

A.2.2. Sim Lifecycle Management - Error codes

A.2.2.1. Update sim status error codes

Table A.3. Update sim status errors

Code	Detail
UPDATE_IN_PROGRESS	A SIM status update request is already pending for this subscription..
STATUS_NOT_ALLOWED	The requested status (<requested status>) is not compatible with the current status (<current status>).
SIM_STATUS_CHANGED	This SIM status has changed since the ticket was created (<new status>). Please re-send your request if needed.
MLM_INTERNAL_ERROR	Internal Malima error.

Table A.4. Error codes and Services

Error	getUpdateSimStatus	submitUpdateSimStatus
MalimaError	X	X
ResultAlready RetrievedError	X	
TooMany Line IdentifiersError		X
UnknownTicketError	X	

A.2.3. Traffic Tracking - Error codes

Table A.5. Error codes and Services

Error	getTrafficTracking	submitTrafficTracking
Invalid Period Error		X
MalimaError	X	X
ResultAlready Retrieved Error	X	
TooMany LineIdentifiers Error		X
Unknown Ticket Error	X	

A.2.4. Session History - Error codes

Table A.6. Error codes and Services

Error	getSessionHistory	submitSessionHistory
Invalid Period Error		X
MalimaError	X	X

Error	getSessionHistory	submitSessionHistory
ResultAlready Retrieved Error	X	
Unknown Line Identifier Error		X
Unknown Ticket Error	X	

A.2.5. Customer Alarms - Error codes

Table A.7. Error codes and Services

Error	create Trigger	delete Trigger	update Trigger	get Alarms	manage Alarms	search Alarms
MalimaError	X	X	X	X	X	X
Trigger Error	X		X			
Trigger Limit Exceeded Error	X					
TooMany AlarmIdentifiers Error				X	X	
TooMany Results Error				X		X
Unknown Customer Environment Number Error	X		X			X
Unknown LineIdentifier Error	X					
Unknown Trigger Identifier Error		X	X			

A.2.6. Device Info - Error codes

Table A.8. Error codes and Services

Error	getSimReport	getStatistics	submitSimSup	submitSimSupLoc
MalimaError	X	X	X	X
ResultAlready Retrieved Error	X			
TooMany LineIdentifiers Error		X	X	X
Unknown Ticket Error	X			

beginner

A.2.7. Incident Diagnostics - Error codes

Table A.9. Error codes and Services

Error	getIncidentDiagnostics
MalimaError	X
networkStatusServiceError	X
geolocServiceError	X

A.2.8. Network Status - Error codes

Table A.10. Error codes and Services

Error	getNetworkStatus
MalimaError	X
networkStatusServiceError	X

A.3. Connectivity

Table A.11. 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.AUTORISATION_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).

beginner

Code	Detail
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.
REQ.REJECTED_BY_SLM_MONITOR	SOAP request refused because the global activity (total number of requests per second) on the web service currently exceeds the pre-defined quota value. In order to avoid this message, you can "smoothe" the activity by adding, in your application, a delay (1 second for instance) between two consecutive requests. (500) SLM Monitor Rejection
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.