**Description of RAPNotifications Java application and WebSocketAPI Version 1.1.**

The **RAPNotifications** java project consists of the following java files:

* WebSocketApi.java
* TestWebSocketClient.java
* WebSocketObservation.java
* Uom.java
* SecResponse.java
* Payload.java
* ObsValue.java
* ObsProperty.java
* Location.java
* KeepAliveThread.java

TestWebSocketClients.java is a java client application testing the WebSocketApi API (WebSocketApi.java) and implements the following tasks:

* Establishes a web socket connection to the remote Resource Access Proxy (RAP) microservice of a platform having platform name = **platformID.** The platform actually is a L2 Cloud platform with a specific interworking interface having value **interWorkingInterface**.
* After a successfully establishment of a web socket connection, the application sends a SUBSCRIPTION COMMAND to RAP and waits to receive notifications of an already registered resource having an internal id = **internalID**.
* The successfully subscription triggers the Resource Access Proxy microservice of L2 Cloud Platform to start sending periodically notifications to this application.
* The Java client application receives periodically notifications of resource’s observation. Parses the notifications with the help of WebSocketObservation.java,Uom.java, Payload.java

SecResponse.java, ObsValue.java, ObsProperty.java and Location.java files.

**Important note**: Before parsing the notification message, the application must call the

***WebScocketApi.checkIfIsObservationMessage();***

If returns false then there is no need to parse the received notification message.

* The application also sends periodically keep alive messages to RAP, with the help of KeepAliveThread Thread.
* The application receives periodically keep alive messages from RAP.
* After a specified duration the application sends an UNSUBSCRIBE COMMAND to RAP and closes the web socket session.

The user of this testing application should use the correct values for the following java variables:

1. ***interWorkingInterface***, is the domain name of the remote L2 Cloud platform where the resources for notifications belong.
2. ***platformID***, is the unique name of the L2 Cloud platform with interworking interface = ***interWorkingInterface*** where the resources for notifications belong.
3. ***InternalID***, is the internal id value of the resource subscribed to send notifications.
4. ***Email, password,*** *are the credentials of the user registered in the ASAPA platform.*

**Description of methods called in TestWebSocketClient.java**

* ***WebSocketApi webSocketApi =***

***new WebSocketApi (String platformID,***

***String interWorkingInterface,***

***String*** ***email,***

***String password);***

The constructor of *WebSocketApi* must be called to assign the parameters ***platformID*, *interWorkingInterface, email*** *and* ***password***. In this step the created WebSocketApi object implements the Web Socket interface offered by the Resource Access Proxy (RAP) of the L2 Cloud platform identified with the ***platformID*** parameter. This constructor is called on receiving the ***NotifyForData*** request as shown in the following diagram and must not be called again for the same **platformID** parameter. The created *WebSocketApi* objectis connected to only one L2 Cloud platform’s RAP identified by ***platformID*** parameter. In case a ***NotifyForData*** request is received for a different ***platformID*** then a new *WebSocketApi* object must be created.

Text

Description automatically generated

* ***String endpointURI = getWebSocketURL();***

The ***getWebSocketURL()***method returns the URI of the Web Socket interface exposed by RAP and has the following form:

***“wss://IIS/rap/notification”****,* where *IIS* is a string value depended on***interWorkingInterface***parameter*.*

* ***String resourceId = getResourceIdFromInternalID(***

***String internalId);***

The ***getResourceIdFromInternalID()***method returns the resource id value of the resource we want to receive notifications. The ***internalId*** parameter is the internal id of the resource.

* ***void addResourceId(String resourceId)***

This method stores the resourceId to a list of resource ids internally created in webSocketApiobject***.***

Must be called after the subscription of the resource.

* ***void removeResourceId(String resourceId)***

This method removes the resourceId from the list of resource ids of webSocketApiobject***.***

Must be called after the unsubscription of the resource.

* ***List<String> getListOfResourceIds()***

This method returns the list of stored resourceIds.

* ***String getMessage(String command, String resourceId)***

This method returns a JSON message used for subscription/ unsubscription of a resource.

The ***command*** argument in case of subscription request is equal to ***WebSocketApi.SUBSCRIBE\_COMMAND*** *string value*.

In case of unsubscribe request the value of ***command*** argument is equal to ***WebSocketApi.UNSUBSCRIBE\_COMMAND*** *string value.*

The ***resourceId*** argument is the resource id of the resource.

* ***void sendMessageToRAP(String message)***

This method sends a request message for subscription/unsubscription to Resource Access Proxy (RAP) microservice.

The string message argument ***message*** is the one created from ***getMessage()***method.

* ***Boolean WebSocketApi.checkIfIsObservationMessage(String message);***

This method is called on reception of notification message from RAP. The application should check if the received ***String message*** is a valid Observation message. If this method returns false then no specific handling of message is needed.

If the ***checkIfIsObservationMessage ()*** method returns true then the ***String message*** is an observation Json message. In this case the application should parse the ***String message*** with the help of the following java classes:

* WebSocketObservation.java
* Uom.java
* SecResponse.java
* Payload.java
* ObsValue.java
* ObsProperty.java
* Location.java

The following code is an example of handling the observation message received:

ObjectMapper objectMapper = new ObjectMapper();  
try {  
 Payload payload = objectMapper.readValue(message,WebSocketObservation.class).PayloadObject;  
 String resultTime = payload.getResultTime();  
 String samplingTime = payload.getSamplingTime();  
 String resourceId = payload.getResourceId();  
 float longitude = payload.getLocation().getLongitude();  
 float latitude = payload.getLocation().getLatitude();  
  
 ArrayList<Object> descriptionList = payload.getLocation().getDescription();  
 ArrayList<ObsValue> obsList = payload.getObsValues();  
  
 for (int i = 0; i < obsList.size(); i++ ){  
 String observedValue = obsList.get(i).getValue();  
 String observedSymbol = obsList.get(i).getUom().getSymbol();  
 System.*out*.println("Property name: " + obsList.get(i).getObsProperty().getName() + " Property description " + obsList.get(i).getObsProperty().getDescription() + " has value : " + observedValue + " " + observedSymbol);  
 }  
} catch (IOException e) {

e.printStackTrace();  
}

Keep alive messages:

The user of the application must create a thread to send keep alive messages periodically to RAP. An example is the following code:

***KeepAliveThread keepAliveThread = new KeepAliveThread ();***

***keepAliveThread.start ();***

* ***void sendKeepAliveMessage ()*** method must be called periodically (<60 seconds) to send keep alive message to RAP.

This method is called inside KeepAliveThread.

When a WebSocketApi object is created, then the following method must be called:

***keepAliveThread.addWebSocketApi(webSocketApi);***

* ***boolean restartSession(WebSocketContainer container, LocalSocketClient localSocketClient);***

This method should be called when the web socket communication with the RAP is interrupted. The ***restartSession***() method establishes again a new web socket link to the RAP and subscribes all the stored resource ids.