Persuasion API - Reference

# Setup Instructions

There is no automated setup process available as of now to install and configure the Persuasion API. Manual installation and configuration of dependent components is required to get the API up and running. The following section explains these steps in detail.

## Pre-requisites

The following software must be installed and configured as directed below before installing the API to a web container.

#### Java v8 or above

Java version 8 or later is required for the Java based Persuasion API. Latest version can be downloaded from <https://java.com/en/download/>. Installation instructions can be found in the download page. After performing the install, make sure that JAVA\_HOME path variable is set to the Java installation directory (not including the bin directory) and that JAVA\_HOME/bin is added to the system path.

#### MySQL Server 5.x or above

MySQL is the database server used in Persuasion API. Latest version of MySQL can be downloaded from <https://www.mysql.com/downloads/>. Download the offering appropriate to your enterprise requirements. Install and configure a user for access by the API with read and write permissions. Installation instructions can be found at <https://dev.mysql.com/doc/refman/5.7/en/installing.html>. If the database is installed in a different server than the API, make sure that the firewall is setup accordingly that the database port (default: 3306) is accessible from the server were API is installed. Also, make sure that the database service is set to start with system startup.

#### Apache ActiveMQ

Apache ActiveMQ broker is the Java Messaging Service (JMS) provider used by the API for internal and external communications. Latest version can be downloaded from <http://activemq.apache.org/download.html>. Installation instructions can be found at <http://activemq.apache.org/installation.html>. If the broker is installed in a different server than the API, make sure that the firewall is setup accordingly that the broker port (default: 8161) is accessible from the server were API is installed. Make sure that the broker is set to start with system startup. If not, add the command ‘*<path\_to\_activemq\_broker\_installation>/bin/activemq start*’ to the startup script.

#### Apache Tomcat

Apache Tomcat is the preferred web container for installing the API services. Any similar web container should serve the purpose. Latest version of Tomcat can be downloaded from <https://tomcat.apache.org/>. Make sure that Tomcat is setup as a service and is set to start at system startup.

## Setting up the database

After MySQL is installed, the Persuasion API database and the associated tables needs to be created. Download the file *persuasionapi\_create\_scripts.sql* and run it against the installed MySQL instance. Create a user with read and write permissions to the *persuasion\_api* database. This will be required later during configuration of the API.

## Installing and configuring the services component

Open Tomcat manager, typically located in http://<server>:8080/manager. Download the file *PersuasionAPI.war* and deploy it in Tomcat. This can be done in the *WAR file to deploy* section of the manager, by selecting the file *PersuasionAPI.war* and clicking on *Deploy* button. Installation can be verified by clicking on */PersuasionAPI* in Tomcat manager to open without any error.

Next, we will let the API know how to access the installed database and the Apache ActiveMQ instance. The configuration file *persuasionapi.properties* for Persuasion API is located in <path\_to\_tomcat\_installation>/webapps/PersuasionAPI/WEB-INF/classes/properties.

* Enter the path to the MySQL database for the field value jdbc.url (typically jdbc://<servername>:<port>/persuasion\_api)
* Enter the configured username and password for the fields jdbc.username and jdbc.password
* Enter the path to the ActiveMQ server for the field activemq.server (typically tcp://<servername>:<port>)
* Restart the tomcat server or click ‘Reload’ for the application /PersuasionAPI in Tomcat manager

## Installing and configuring the background processor

Similar to the services component, the background processor is also managed by a web container such as Tomcat. Deploy the file PersuasionAPIAsyncProcessor.war in Tomcat manager and verify the installation. Next follow the same steps as above to configure /PersuasionAPIAsyncProcessor. The configuration file will be found in <path\_to\_tomcat\_installation>/webapps/PersuasionAPIAsyncProcessor/WEB-INF/classes/properties. Reload the application or restart tomcat for the changes to take effect.

# Developing with Persuasion API

Once the above setup is done and the API is up and running, integration with a consumer is simple. The clients have to do two things.

## Creating/maintaining configurations

<Add content here after creating the UI>

## Consuming the API services

Java based consumers can import the PersuasionAPIClient package and start consuming the API through the available service proxies. Details of the request/response parameters can be found in the [API Reference](#_API_Reference) section of this document.

Important: The PersuasionAPIClient package has to be initialized before it can be used. The context path URL to the installed Persuasion API services component PersuasionAPI is required to do this. This can be done by creating the initialize() method on osu.ceti.persuasionapi.PersuasionAPIClient. Example URL: https://<server>:8080/PersuasionAPI. It is important to specify this without the slash(/) at the end of the URL. An example code for retrieving all badge assignments for a user is shown below.

PersuasionAPIClient client = **new** PersuasionAPIClient();

client.initialize("http://localhost:8080/PersuasiveAPI");

String userId = "2";

BadgeService badgeService = **new** BadgeService();

List<GetUserBadgeResponse> response = badgeService.getAllBadgesForUser(userId);

//do something with the response

PersuasionAPIClient proxy details for each service are specified in the [API Reference](#_API_Reference) section.<Add this>. If the service returns a failure, or in case of an error in calling the service, an exception of type osu.ceti.persuasionapi.core.exceptions.PersuasionAPIException is thrown. This contains the error code and error message for more detailed information.

Non-Java based consumers can call the REST based APIs directly using the method of their choice. Details of the API methods can be found in the [API Reference](#_API_Reference) section of this document.

## Consuming messages from JMS queues

All the JMS queues used by the API are located in the same Apache ActiveMQ server configured during the initial setup of the API. ActiveMQ has platform specific adapter that can be used to connect to the server to access the queues. Refer to ActiveMQ documentation at <http://activemq.apache.org/connectivity.html> for more information.

PersuasionAPIClient package contains abstract listener interfaces that can help Java based consumers to easily implement message listeners.

To implement a listener for a particular queue, create a custom listener class extending the abstract implementation osu.ceti.persuasionapi.client.MessageListener and provide the implementation for the method handleMessage(). Then register this new listener class as the listener for the queue and start listening. An example implementation for listening to queue *com.example.queue1* is shown below.

String queueName = "com.example.queue1";

MessageListenerRegistrar.registerListener(queueName, ExampleMessageListener.**class**);

MessageListener listener = MessageListenerRegistrar.getListener(queueName);

listener.start();

Listening can be stopped and started at any time by using the *stop()* and *start()* methods.

MessageListener listener = MessageListenerRegistrar.*getListener*(queueName);

listener.stop();

//do some processing

listener.start();

The listener can also be permanently unregistered and removed from the configuration as below.

MessageListenerRegistrar.unregisterListener(queueName);

Note that only one listener can be registered for a queue at any point in time. And hence an attempt to register more than one listener will result in an error. Also, a listener implementation can be used to listen to only one queue and an attempt to register it to multiple queues will result in an error.

# API Reference

The services provided by Persuasion API are REST based POST services. They use JSON for data transfer. PersuasionAPIClient contains proxy implementations for these services.

The general format for a request is:

|  |
| --- |
| {  "data":  {  "field1":"value1",  "field2":"value2"  }  } |

Request data specified by the service methods are to be wrapped inside the object *data* in the JSON request. If a particular field is a list or an object, it can be nested using standard JSON formats.

In case of a single request parameter, the value of the parameter can be directly specified under *data*. The field data is then considered as a string key-value pair and not an object.

|  |
| --- |
| {  "data":<value>  } |

The general response format in case of a success is:

|  |
| --- |
| {  "responseType": "SUCCESS",  "data":  {  "field1": "value1",  "field2":"value2"  }  } |

The response type will contain *SUCCESS* and the response content will be wrapped inside *data*. In the case where there are no response parameters, only *responseType SUCCESS* will be returned in the response.

The general response format in case of an error is:

|  |
| --- |
| {  "responseType": "FAILURE",  "errorCode": <error\_code>,  "errorMessage": <error\_message>  } |

The response type will contain *FAILURE* and the error code and error message will contain appropriate details.

## Activity/report

This API is used for reporting user activity to the API. This can be accessed using the *activity/report* context path i.e. http://<server>:<port>/PersuasionAPI/activity/report.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| userId | ID of the user in the consuming system | String |
| activityName | A name the uniquely identifies the activity | String |
| value | A value corresponding to the activity that can be helpful in rule processing | String |

#### Response Parameters

There are no special response parameters.

#### PersuasionAPIClient Proxy

This service can also be accessed using the PersuasionAPI Client proxy method osu.ceti.persuasionapi.services.ActivityService:reportActivity(). It takes plain string input of all the three request parameters as above.

ActivityService activityService = **new** ActivityService();

activityService.reportActivity(userId, activityName, activityValue);

//No further processing as no result is expected from the service

Remember that PersuasionAPIClient needs to be initialized before consuming services. In case of a failure in calling the service, a PersuasionAPIException is thrown as mentioned already.

## Badges/getUserBadgeForClass

This API is used to retrieve the badge details assigned for the user for a particular badge class. This can be accessed using the *badges/getUserBadgeForClass* context path.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| userId | ID of the user in the consuming system | String |
| badgeClass | Class of the badge for which badge assignment is searched for | String |

#### Response Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Type** |
| userId | ID of the user in the consuming system | String |
| badgeClass | Class of the badge for which badge assignment is searched for | String |
| badgeLevel | Level of the badge that is assigned for the user in this badge class | Integer |
| badgeName | Name of the badge as configured | String |
| badgeDesc | Description of the badge as configured | String |
| Image | URL to the image configured for this badge | String |

#### PersuasionAPIClient Proxy

This service can also be accessed using the PersuasionAPI Client proxy method osu.ceti.persuasionapi.services.BadgeService:getUserBadge(). It takes plain string input of all the three request parameters as above.

BadgeService badgeService = **new** BadgeService();

GetUserBadgeResponse response = badgeService.getUserBadge(userId, badgeClass);

//do something with the response

The proxy returns the response in a wrapper object of type osu.ceti.persuasionapi.services.wrappers.GetUserBadgeResponse.

## Badges/getAllBadgesForUser

This service provides the current list of badges from all badge classes assigned to the user. This can be accessed using the *badges/getAllBadgesForUser* context path.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| User ID(data) | ID of the user in the consuming system | String |

Note that this service method expects only one request parameter. In this case, the format specified for single request parameter above applies i.e. specify the User ID value directly under *data* field of the request.

#### Response Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Type** |
| userId | ID of the user in the consuming system | String |
| badgeClass | Class of the badge for which badge assignment is searched for | String |
| badgeLevel | Level of the badge that is assigned for the user in this badge class | Integer |
| badgeName | Name of the badge as configured | String |
| badgeDesc | Description of the badge as configured | String |
| Image | URL to the image configured for this badge | String |

Note that this returns a list of badges, each containing the set of fields as above. In this case, the *data* field in the JSON response will be treated as a list and will return multiple sets of values.

#### PersuasionAPIClient Proxy

This service can also be accessed using the PersuasionAPI Client proxy method osu.ceti.persuasionapi.services.BadgeService:getUserBadge(). It takes plain string input of all the three request parameters as above.

BadgeService badgeService = **new** BadgeService();

List<GetUserBadgeResponse> response = badgeService.getAllBadgesForUser(userId);

//do something with the response

The proxy returns the list of all user badges in a wrapper object list of type osu.ceti.persuasionapi.services.wrappers.GetUserBadgeResponse.

## User/attribute/update

This service enables a way to create/update attribute values for individual users. This can be accessed using the *user/attribute/update* context path.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| userId | ID of the user in the consuming system | String |
| attributes | List of user attribute pairs of the composite type UserAttribute | List |

*Composite type UserAttribute*

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| attributeName | A name that uniquely identifies the attribute | String |
| value | Value to be assigned to the user for this attribute | String |

#### Response Parameters

There are no special response parameters.

#### PersuasionAPIClient Proxy

Two variants of the proxies are available for this service – one to update only one attribute for a user and the other to update multiple attributes for a user.

Single attribute update can be performed by using updateUserAttribute() in osu.ceti.persuasionapi.services.UserAttributeService. It takes as input a user ID, an attribute name and a corresponding attribute value. Multiple attribute update can be performed using updateMultipleAttributeForUser(). It takes as input a user ID and a key-value pair map of the user attributes to be modified.

UserAttributeService userAttributeService = **new** UserAttributeService();

userAttributeService.updateUserAttribute(userId, attributeName, value);

//No further processing as no result is expected from the service

## User/attribute/getUserAttributeValue

This service retrieves a single attribute value for the given user. This can be accessed using the *user/attribute/getUserAttributeValue* context path.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| userId | ID of the user in the consuming system | String |
| attributeName | Name of the attribute for which value is to be retrieved | String |

#### Response Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Type** |
| userId | ID of the user in the consuming system | String |
| attributeName | Name of the attribute for which value is to be retrieved | String |
| value | Value of the attribute | String |

#### PersuasionAPIClient Proxy

This service can be accessed using the proxy method getAttributeValueForUser() in osu.ceti.persuasionapi.services.UserAttributeService. It takes plain string input values as specified above.

UserAttributeService userAttributeService = **new** UserAttributeService();

GetUserAttributeResponse response =

userAttributeService.getAttributeValueForUser("1", "user\_type");

//do something with the response

## User/social/getNotificationsAfterTime

This service retrieves all social notifications generated by the user after the given time. This can be accessed using the user/social/getNotificationsAfterTime context path.

#### Request Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| userId | ID of the user in the consuming system | String |
| timestamp | Time stamp after which the notifications are to be retrieved | String of the format yyyy-MM-dd HH:mm:ss |

#### Response Parameters

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Type** |
| userId | ID of the user in the consuming system | String |
| notifications | List of social notifications generated by the user of the composite type SocialNotification | List |

*Composite Type SocialNotification*

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Type** |
| notificationText | The actual notification content | String |
| timestamp | Time when the notification was generated | String of the format yyyy-MM-dd HH:mm:ss |

#### PersuasionAPIClient Proxy

This service can be accessed using the proxy method getNotificationsAfterTime() in osu.ceti.persuasionapi.services.UserSocialNotificationService.

UserSocialNotificationService socialService = **new** UserSocialNotificationService();

GetSocialNotificationsResponse response =

socialService.getNotificationsAfterTime(userId, timeStamp);

//do something with the response

In case the user is not found or no notifications are found, an empty list of notifications is returned in a success response.

## Consuming Email notifications

Email notifications configured to be sent to the users in Persuasion API are queued up in a JMS queue for the consuming systems to pick up and send according to the user’s communication preferences. These are posted to the same Apache ActiveMQ server configured during the setup of the API.

There are two queues for email notifications – one for the messages generated by badge assignments and the other for the messages generated by rule actions.

Badge assignments posts messages to: *osu.ceti.persuasionapi.badgenotifications.email*

Rule actions posts messages to: *osu.ceti.persuasionapi.rulenotifications.email*

The JMS messages contains the following information fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Accepts** |
| user\_id | ID of the user for whom the email message is for | String |
| subject | Email subject | String |
| body | Email body | String |

These queues can be accessed by the method mentioned in [Consuming messages from JMS queues](#_Consuming_messages_from).

## Consuming Custom JMS messages

Custom JMS notifications are posted to the configured queues in the same Apache ActiveMQ server configured during the setup of the API. These JMS messages contains key value pairs of all activity and attribute values used for processing the rule that generates these messages. These can be accessed by the method mentioned in [Consuming messages from JMS queues](#_Consuming_messages_from).