Hewlett Packard Enterprise

AutoPass License Server

API Document

Revised: April 2023

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.
Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.
UNIX® is a registered trademark of The Open Group.
This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

About this PDF Version of User Guide

This document is a PDF version of the User Guide. This PDF file is provided so you can easily print multiple topics from the help information or read the User Guide in PDF format. Because this content was originally created to be viewed as User Guide in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the User Guide.

Table of Contents

Introduction

The APLS APIs allows user to perform few restricted operations on license usage report and reservation pool members.

APLS APIs are built using REST principles and follows HTTP rules, enabling HTTP clients to interact with the APIs.

Every resource is exposed as URL. The URL of each resource can be obtained by accessing the API Root Endpoint.

https://<hostname>:<port>/autopass/wsservices/v9.3

Also a new API is to retrieve the licenses associated with a license token checked out by client is exposed with

https://<hostname>:<port>/autopass/wsservices/v9.6

Note: the URL configuration will modify based on the APLS server deployment configurations.

Default is protocol and port is *https* and *5814.*

Getting Started

All APLS APIs requires a minimum of one mandatory header.

- **Authorization**: Authentication request header.

For Example (Linux):

```
$ curl https://iwfvm07739.hpeswlab.net:5814/autopass/wsservices/v9.3/reservation/pool/members?poolName=”Development”
    -H 'Authorization: Basic YXBsc1VzZXI6cGFzc3dvcmQ='
```

Authentication

The APIs use “Basic” authentication to authenticate and authorize users to perform operations on APLS APIs.

Default view only user configured in APLS is **‘aplsUser’: “password”** (*YXBsc1VzZXI6cGFzc3dvcmQ=)*

To manage users in APLS refer **“User Management”** section in user guide.

To pass the credentials to the rest API use the below format:

- <user name>:<password>
- Encode the above value using *Base64* encoding (e.g., *YXBsc1VzZXI6cGFzc3dvcmQ=*)
- Append encoded value with “Basic“, e.g. “Basic *YXBsc1VzZXI6cGFzc3dvcmQ=”*
- Add above as “Authorization” header value

Points to Note

- Authentication token should be generated based on the user managed in APLS. If the user is deleted, the APIs will not able to authenticate and authorize the token and you will receive *401 error code*
- To add or delete operations, the authenticated token should belong to an administrator user in APLS.

- In case of SSL connection issue with *curl* command, search for “Curl disable certificate verification” to allow SSL connection without validating the server certificate. Please note the curl command given above is just a sample and meant for testing purpose only. It is always recommended that client implement code to validate the server certificate before proceeding with the next step.

HTTP Methods

Using **GET** method, user can get the list of resources or details of a particular instance of a resource.

To get a list of pool member

```
$ curl https://iwfvm07739.hpeswlab.net:5814/autopass/wsservices/v9.3/reservation/pool/members?poolName=”Develpoment”
    -H 'Authorization: Basic YXBsc1VzZXI6cGFzc3dvcmQ='
```

Currently the below three methods are used in API queries.

Find the below table for reference:

Method	Description
GET	**To retrieve resources.**
POST	**To create resources and performing resource actions.**
DELETE	**To delete resources.**

Response

Response, by default in the XML format.

Find the below table for reference:

NODE NAME	Description
Schema	**Each response schema differs based on the resources**
Status	**Each API call will return an status with *SUCCESS/ PARTIAL SUCCESS/FAIL***
errorCode	**Custom error code from APLS server**
errorMessage	**Generic error message**
customMessage	**Custom error message specifying the nature of the failure**

Sample response structure,

```
<ReservationPoolResult>
   <name>Development</name>
   <status>SUCCESS</status>
   <members>
      <userNameList>
         <userName>ASIAPACIFIC/rpadmava</userName>
      </userNameList>
      <ipAddressList/>
      <hostList/>
      <clientIDList/>
   </members>
</ReservationPoolResult>
```

Other Format Support

All GET APIs supports *JSON* format. To return the response in JSON format Accept head as well for which the required response format need to be specified in the respective request's Accept header.

For example:

Pass **Accept** header as **application/json** below result displays


```
{
   "name": "Development",
   "status": "SUCCESS",
   "members":    {
      "userNameList": {"userName": ["ASIAPACIFIC/rpadmava"]},
      "ipAddressList": {"ipAddress": []},
      "hostList": {"host": []},
      "clientIDList": {"clientID": []}
   }
}
```


Errors

APLS APIs uses HTTP status codes to indicate success or failure of an API call. In general, status codes in the *2xx* range means 'success', *4xx* range means there was an 'error' in the provided information, and those in the *5xx* range indicates 'server side errors'.

Commonly used HTTP status codes by APLS are listed below:

Status Code	Description
200	**Ok**
201	**Created**
401	**Unauthorized (Invalid AuthToken)**
403	**Forbidden**
404	**URL Not Found**
406	**Not Acceptable**
500	**Internal Error**

Reservation Pool Users

Member Attributes

Member attributes should be passed when user call the API with *POST or DELETE methods* which contains *four attributes* data. Refer user guide for more details (**"Client User Management"**)

Refer the below table to understand member attributes for a given reservation pool:

ATTRIBUTE	DESCRIPTION
userName	**The Windows or Unix user name of a client user *DOMAIN/USERNAME*** **For eample: ASIAPACIFIC/rpadmava**
ipAddress	**The IP address of the system from where the client accesses the AutoPass License Server.** **For eample: 16.168.213.40**
host	**The host address of the client system** **For example: *RPADMAVA2***
clientID	**A unique value configured for each client based on the product's support for this attribute.** **For example: RPADMAVA2 (any configured string from the client)**

Point to Note:

- The above four attributes can be encompassed as *<userNameList>, <ipAddressList>,<hostList>, </clientIDList>*, these lists are grouped under **"<members>"**
- The above attributes will be used to "Add/Delete/Get" member details from a reservation pool

Create Reservation Pool Members

This REST API will allow to add a list of members to already created reservation pool.

Adding members to APLS reservation pool can be done in two ways,

- By signing in *APLS UI as an administrator*. For more details how to create a pool and add users to a client pool please refer user guide (**"How to Manage Client User Access"**)
- By calling REST API by issuing the *HTTP POST* request to the *APLS endpoint/handler*. Please read the following details for more information about how to pass *POST request*.

Important information

Create a reservation pool through APLS UI in order for this rest service to successfully add the members. Please refer the user guide (**"Add a user pool"**).

Request

REQUEST

Method	URL
POST	https://<hostname>:<port>/autopass/wsservices/v9.3/reservation/pool/members

Request Examples:

- Use case: Add a list of domain users name to a pool.

```
<ReservationPool>
     <name>Development</name>
     <members>
            <userNameList>
                 <userName>ASIAPACIFIC/rpadmava</userName>
                 <userName>ASIAPACIFIC/ramana</userName>
                 <userName>ASIAPACIFIC/anantha</userName>
            </userNameList>
   </members>
</ReservationPool>
```

The above request contains the media type “application/xml” as an input.

The API is called with the above xml as an input with **HTTP POST** method to create the list of members for the pool.

You can also use combination of userName, ipAddress, host and clientID can be used to add to the reservation pool.

For example:

```
<ReservationPool>
     <name>Development</name>
     <members>
            <userNameList>
                 <userName>ASIAPACIFIC/rpadmava</userName>
                 <userName>ASIAPACIFIC/ramana</userName>
            </userNameList>
            <ipAddressList>
                 <ipAddress>16.168.213.40</ipAddress>
                 <ipAddress>16.168.213.41</ipAddress>
            </ipAddressList>
            <hostList>
                 <host>RPADMAVA2</host>
                 <host>RAMAN</host>
            </hostList>
            <clientIDList>
                 <clientID>RPADMAVA2</clientID>
                 <clientID>RAMANA2</clientID>
            </clientIDList>
   </members>
</ReservationPool>
```

To understand when we need to add alist of various other information, please check the “Client User Management” section in the APLS user guide.

Response

The Response by default will be in *XML* format. The status of this call can be identified through *status* field of the response object which will contain SUCCESS / PARTIAL SUCCESS / FAIL.

If the API request to add members are successful, the members will be added to the specified pool in APLS *UI ->Reservation Management.*

STATUS	RESPONSE

201 Created	**Below is the response when the POST call to add members to a pool is successful.**
	
	`<ReservationPoolResult>`
	`<name>Development</name>`
	`<status>SUCCESS</status>`
	`<summary>`
	`<numberOfUsersUpdated>3/3</numberOfUsersUpdated>`
	`</summary>`
	`</ReservationPoolResult>`

200 OK	**In case the second query is executed immediately after the first one, you will get the following response**
	
	`<ReservationPoolResult>`
	`<name>Development</name>`
	`<status>PARTIAL SUCCESS</status>`
	`<summary>`
	`<numberOfUsersUpdated>0/2</numberOfUsersUpdated>`
	`<numberOfIPsUpdated>2/2</numberOfIPsUpdated>`
	`<numberOfHostsUpdated>2/2</numberOfHostsUpdated>`
	`<numberOfClientIDsUpdated>2/2</numberOfClientIDsUpdated>`
	`</summary>`
	`<errorList>`
	`<errorDetail>`
	`<errorCode>12002</errorCode>`
	`<errorMessage>Member already exist</errorMessage>`
	`<customMessage>User by (ASIAPACIFIC/rpadmava) already exists.</customMessage>`
	`</errorDetail>`
	`<errorDetail>`
	`<errorCode>12002</errorCode>`
	`<errorMessage>Member already exist</errorMessage>`
	`<customMessage>User by (ASIAPACIFIC/ramana) already exists.</customMessage>`
	`</errorDetail>`
	`</errorList>`
	`</ReservationPoolResult>`

404 NOT FOUND	**In case the pool does not exist in APLS or url is malformed**
	
	`<ReservationPoolResult>`
	`<status>FAIL</status>`
	`<errorList>`
	`<errorDetail>`
	`<errorCode>12001</errorCode>`
	`<errorMessage>Pool Name doesn't exists.</errorMessage>`
	`<customMessage>Pool by name({QA})doesn't exist at License Server</customMessage>`
	`</errorDetail>`
	`</errorList>`
	`</ReservationPoolResult>`

400 BAD REQUEST	**In case the input is malformed or not accepted, you will get 400 Bad Request from the server**
500 INTERNAL ERROR	**In case of APLS server is not reachable/down or any internal issue.**

Points to note

- It is recommended to restrict the maximum number of members to be added to 100, for faster response.
- Currently the request input support the media type “application/xml” only. The response media can be either an xml or json based on the “Accept” header configuration in your query.
- Also only APLS admin users are authorized to execute this query, please ensure you pass the “Authorization” for this query using the admin users in APLS.
- Pool name is case sensitive.

Retrieve Pool Member List

This API will get the member values such as *User Name, IP Address, Host ID and Client ID* for a given Pool name.

Request

REQUEST

Method	**URL**
GET	**https://<hostname>:<port>/autopass/wsservices /v9.3/reservation/pool/members?poolName={pool_name}**

@**param** poolName**:**
Reservation pool as configured in Reservation Management -> Pool Management of AutoPass License Server. For more details to get the {pool_name}, please refer the user guide **("Pool Management Tab").**

Response

Response by default will be in the *XML format* which contains *HTTP* status code and member values for a given reservation pool name. Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	**RESPONSE**
200 OK	Below is the **success response** for the above query. Input value for {pool_name} in this API query is ‘Development’(*case sensitive*)

```
<ReservationPoolResult>
   <name>Development</name>
   <status>SUCCESS</status>
   <members>
      <userNameList>
         <userName>ASIAPACIFIC/rpadmava</userName>
      </userNameList>
```

	<ipAddressList/> <hostList/> <clientIDList/> </members> </ReservationPoolResult>
200 OK	**In case there is no members configured for the reservation pool empty lists are returned as response** <ReservationPoolResult> <name>Development</name> <status>SUCCESS</status> <members> <userNameList/> <ipAddressList/> <hostList/> <clientIDList/> </members> </ReservationPoolResult>
404 NOT FOUND	**Below response if the pool name is incorrect (Pool name does not exist)** <ReservationPoolResult> <status>FAIL</status> <errorList> <errorDetail> <errorCode>12001</errorCode> <errorMessage>Pool Name doesn't exists.</errorMessage> <customMessage>Pool by name ({quality assurance}) doesn't exist at Licens e Server</customMessage> </errorDetail> </errorList> </ReservationPoolResult>
400 BAD REQUEST	**In case the input is malformed or not accepted, you will get 400 Bad Request from the server**
500 INTERNAL ERROR	**In case of APLS server is not reachable/down or any internal issue.**

Points to note

- The other format supported for response is JSON
- Pool name is case sensitive.

Delete Reservation Pool Members

This REST API will allow you to DELETE a list of member from a reservation pool.

Create a *HTTP DELETE* request to delete the attribute values from a given Pool Name. Authentication details must be also submitted as part of *HTTP* request header.

Deletion of members from APLS reservation pool can be done in two ways,

- By signing in to the *APLS UI*. For more details how to remove client user attributes from pool. Please refer user guide (**"How to Manage Client User Access"**)
- By calling the API by issuing the *HTTP DELETE* request to the *APLS* endpoint/handler.

By calling REST API by issuing the *HTTP POST* request to the *APLS endpoint/handler*. Please read the following details for more information about how to pass *POST request*.

Please read the following details for more information about how to pass DELETE request.

Important information

Ensure a reservation pool exists through APLS UI in order for this rest service to successfully delete the members. Please refer the user guide (**"Remove a client user's attribute from a pool"**).

The following is the End Point to be used with *HTTP DELETE* request. The request shall be in *XML* format.

Request

REQUEST

Method	URL
DELETE	https://<hostname>:<port>/autopass/wsservices/v9.3/reservation/pool/members

Sample Requests:

Use case remove list of domain users from a pool

```
<ReservationPool>
     <name>Development</name>
     <members>
            <userNameList>
                 <userName>ASIAPACIFIC/rpadmava</userName>
                 <userName>ASIAPACIFIC/ramana</userName>
                 <userName>ASIAPACIFIC/anantha</userName>
            </userNameList>
   </members>
</ReservationPool>
```

The abve request contains the media type “application/xml” as an input.

The API is called with the above xml as an input with **HTTP DELETE** method to delete the list of members for the pool.

You can also use combination of userName, ipAddress, host and clientID can be used to add to the reservation pool.

For example:

```
<ReservationPool>
     <name>Development</name>
     <members>
            <userNameList>
                 <userName>ASIAPACIFIC/rpadmava</userName>
                 <userName>ASIAPACIFIC/ramana</userName>
            </userNameList>
            <ipAddressList>
                 <ipAddress>16.168.213.40</ipAddress>
                 <ipAddress>16.168.213.41</ipAddress>
            </ipAddressList>
            <hostList>
                 <host>RPADMAVA2</host>
                 <host>RAMAN</host>
            </hostList>
            <clientIDList>
                 <clientID>RPADMAVA2</clientID>
                 <clientID>RAMANA2</clientID>
            </clientIDList>
   </members>
</ReservationPool>
```

Response

The Response by default will be in *XML* format which contain a *HTTP* status code and response XML.

Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	RESPONSE

200 OK	Below is the response when the DELETE call to remove members from a pool is successful
	`<ReservationPoolResult>`
	`<name>Development</name>`
	`<status>SUCCESS</status>`
	`<summary>`
	`<numberOfUsersUpdated>3/3</numberOfUsersUpdated>`
	`</summary>`
	`</ReservationPoolResult>`

200 OK	**In case the second query is executed immediately after the first one, you will get the following response**
	`<ReservationPoolResult>`
	`<name>Development</name>`
	`<status>PARTIAL SUCCESS</status>`
	`<summary>`
	`<numberOfUsersUpdated>0/2</numberOfUsersUpdated>`
	`<numberOfIPsUpdated>2/2</numberOfIPsUpdated>`
	`<numberOfHostsUpdated>2/2</numberOfHostsUpdated>`
	`<numberOfClientIDsUpdated>2/2</numberOfClientIDsUpdated>`
	`</summary>`

	<errorList> <errorDetail> <errorCode>12005</errorCode> <errorMessage>Member doent not exist</errorMessage> <customMessage>User by (ASIAPACIFIC/rpadmava) not exist in (Development)</customMessage> </errorDetail> <errorDetail> <errorCode>12005</errorCode> <errorMessage>Member doent not exist</errorMessage> <customMessage>User by (ASIAPACIFIC/ramana) not exist in (Development)</customMessage> </errorDetail> </errorList> </ReservationPoolResult></ReservationPoolResult>
404 NOT FOUND	**In case the pool does not exist in APLS** <ReservationPoolResult> <status>FAIL</status> <errorList> <errorDetail> <errorCode>12001</errorCode> <errorMessage>Pool Name doesn't exists.</errorMessage> <customMessage>Pool by name({QA})doesn't exist at License Server</customMessage> </errorDetail> </errorList> </ReservationPoolResult>
400 BAD REQUEST	**In case the input is malformed or not accepted, you will get 400 Bad Request from the server**
500 INTERNAL ERROR	**In case of APLS server is not reachable/down or any internal issue. Or a wrong URL is configured.**

Points to note

- It is recommended to restrict the total number of member values deleted to 100 for faster response.
- Currently the request input support the media type “application/xml” only. The response media can be either an xml or json based on the “Accept” header configuration in your query.
- Also only APLS admin users are authorized to execute this query, please ensure you pass the “Authorization” for this query using the admin users in APLS.
- Pool name is case sensitive.

License Usage

ATTRIBUTE	DESCRIPTION
productId	Products code of the product configured in the APLS For example: HP UFT
productVersion	Product's version For example: 12.52
pdfID	Product definition file ID. Internal purpose, please ignore this value.
pdfVersion	Product definition file version. Internal purpose, please ignore this value.
usageStartTimeUTCInSeconds	Start time of the duration for which usage is retrieved. Time is represented as seconds using the standard base time know as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400
usageEndTimeUTCInSeconds	End time for the duration for which usage is retrieved.
featureId	Feature ID
featureVersion	Feature Version
featureType	Feature Type (FLOATING)
featureDescription	Feature Description
peakUsage	Peak usage during the duration
averageUsage	Average usage for the duration. Please *notice the web service calculates,* Average = Total capacity checked out for the duration / Number of check outs. In case of UI, the average calculated is per day and for the duration specified it is represend as, Average = ∑(Average(day(1)) …Average(day(n)) / number of days
totalCapacityCheckedout	Total capacity checked out for the duration
applicationName	Application name.
poolName	Reservation pool name
poolCreatedBy	Reservation pool created by

poolCreationTime	**Time the reservation pool created**
allotedCapacityInPool	**Reserved capacity of the pool**
isRestricted	**Is the members restricted to the pool's reserved capacity**

Points to note:

- The above attributes are encompassed such as PoolFeatureUsage, ApplicationUsage, and ProductFeaturePeakUsage in case of XML media type, based on the REST API

Retrieve Feature Usage

This REST API allows a developer to retrieve the usage for a Feature.

Request

REQUEST

Method	**URL**
GET	**https://<hostname>:<port>/autopass/wsservices/v9.3/usage/feature?featureId={feature_id}&featureVersion={feature_version}&startTime={startTimeInEpochSeconds}&endTime={startTimeInEpochSeconds}**

@**param** featureId:
Identifier of the feature for which usage is to be retrieved. Please refer to "License Usage" pane in APLS to identify the {feature_id} of the feature

@**param** featureVersion:
Version of the feature for which the usage is to be retrieved. Please refer the same "License Usage" pane in APLS to identify the {feature_version} of the feature

@**param** startTime:
Start time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time known as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

@**param** endTime:
End time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time known as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

Points to note:
By default, the response mediat type is of XML formation; however all GET APIs supports *JSON* format as well. To generate JSON the request's Accept header should have application/json.

Response

Response by default will be in the *XML format* which contains *HTTP* status code and usage for a given feature and duration.

Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	RESPONSE
200 OK	Below is the **success response** for the above query. Input value for {feature_id} is 10616, {feature_version} is 1, {startTimeInEpochSeconds} is 1475519400 and {startTimeInEpochSeconds} is 1475565690.

```
<ProductFeaturePeakUsage>
        <productId>HP</productId>
        <productVersion>12.52</productVersion>
        <pdfID>10027</pdfID>
        <pdfVersion>1.0</pdfVersion>
        <usageStartTimeUTCInSeconds>1475519400</usageStartTimeUTCInSeconds>
        <usageEndTimeUTCInSeconds>1475565690</usageEndTimeUTCInSeconds>
        <featureBasedUsage>
                <feature>
                        <featureId>10616</featureId>
                        <featureVersion>1</featureVersion>
                        <featureType>FLOATING</featureType>
                        <featureDescription>QuickTest Pro Java Add-in Concurrent
User</featureDescription>
                </feature>
                <capacityUsage>
                        <peakUsage>9</peakUsage>
                        <averageUsage>3</averageUsage>
                </capacityUsage>
                <checkoutsUsage>
                        <totalCapacityCheckedout>729</totalCapacityCheckedout>
                </checkoutsUsage>
        </featureBasedUsage>
         <status>SUCCESS</status>
</ProductFeaturePeakUsage
```

404 Not Found — **In case an invalid or unavailable feature's usage is queried**

```
<ProductFeaturePeakUsage>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12012</errorCode>
      <errorMessage>Product not found.</errorMessage>
      <customMessage>Product for feature 1061699:1 is not
configured in the License Server
      </customMessage>
    </errorDetail>
</ProductFeaturePeakUsage>
```

200 OK — **In case of valid feature but usage in not available for the duration**

```
<ProductFeaturePeakUsage>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12005</errorCode>
      <errorMessage>Resource does not exist</errorMessage>
      <customMessage>Usage for resource 10594:1 is not found.</customMess
age>
   </errorDetail>
</ProductFeaturePeakUsage>
```

Points to note

- If client time out is set to 60 seconds and there are more than 1500 license transactions during the specified period, the response may timeout. Please ensure you increase the connection and read timout in this case

Retrieve Reservation Pool Usage

This REST API allows a developer to retrieve the usage for a reservation pool.

Request

REQUEST

Method	URL
GET	**https://<hostname>:<port>/autopass/wsservices/v9.3/usage/pool?poolName ={pool_name}&startTime={startTimeInEpochSeconds}&endTime={startTimeInEpochSeconds}**

@**param** poolName:
Reservation pool as configured in Reservation Management -> Pool Management of AutoPass License Server. For more details to get the {pool_name} information, please refer the user guide **("Pool Management Tab").** *Pool name is case sensitive*. To retrieve usage for the common pool, specify "*Common Pool*" (case insensitive)

@**param** startTime:
Start time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time know as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

@**param** endTime:
End time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time know as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

Points to note:
By default, the response mediat type is of XML formation; however all GET APIs supports *JSON* format as well. To generate JSON the request's Accept header should have application/json.

Response

Response by default will be in the *XML format* which contains *HTTP* status code and usage for a given feature and duration.

Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	RESPONSE
200 OK	Below is the **success response** for the above query. Input value for {pool_name} is Development, {startTimeInEpochSeconds} is 1475519400 and {startTimeInEpochSeconds} is 1475565690.

```
<PoolFeatureUsage>
        <poolName>Development</poolName>
        <usageStartTimeUTCInSeconds>1473967740</usageStartTimeUTCInSeconds>
        <usageEndTimeUTCInSeconds>1475561488</usageEndTimeUTCInSeconds>
        <featureUsageDetails>
           <poolUsage>
             <poolFeatureDetails>
                <featureId>10606</featureId>
                <featureVersion>1</featureVersion>
```

```
                <featureType>FLOATING</featureType>
                <featureDescription>QuickTest Professional Core Concurrent User</
featureDescription>
                <allotedCapacityInPool>500</allotedCapacityInPool>
                <isRestricted>false</isRestricted>
            </poolFeatureDetails>
                <capacityUsage>
                  <peakUsage>243</peakUsage>
                  <averageUsage>2</averageUsage>
                </capacityUsage>
                <checkoutsUsage>
                    <totalCapacityCheckedout>2636</totalCapacityCheckedout>
                </checkoutsUsage>
          </poolUsage>
        </featureUsageDetails>
        <status>SUCCESS</status>
</PoolFeatureUsage>
```

404 Not Found — **In case an invalid or unavailable pool usage is queried**

```
<PoolFeatureUsage>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12005</errorCode>
      <errorMessage>Resource does not exist</errorMessage>
      <customMessage>Resource Development1 does not exist.</customMessag>
   </errorDetail>
</PoolFeatureUsage>
```

200 OK — **In case of valid pool name but usage in not available for the duration**

```
<PoolFeatureUsage>
   <poolName>Development</poolName>
   <poolCreatedBy>admin</poolCreatedBy>
   <poolCreationTime>1475668559</poolCreationTime>
   <usageStartTimeUTCInSeconds>1475692200</usageStartTimeUTCInSeconds>
   <usageEndTimeUTCInSeconds>1475778599</usageEndTimeUTCInSeconds>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12005</errorCode>
      <errorMessage>Resource does not exist</errorMessage>
      <customMessage>Usage for resource Development is not found.</custom
Message>
   </errorDetail>
</PoolFeatureUsage>
```

Points to note:

- Single pool usage can be retrieved through one request.
- Pool name other than “Common Pool” is case sensitive.
- If client time out is set to 60 seconds and there are more than 1500 license transactions during the specified period, the response may timeout. Please ensure you increase the connection and read timout in this case

Retrieve Application Usage

This REST API allows a developer to retrieve usage against the feature which have been checked out for the given application name from the client.

Request

REQUEST

Method	URL
GET	https://<hostname>:<port>/autopass/wsservices/v9.3/usage/application?applicationName={application_name}&startTime={startTimeInEpochSeconds}&endTime={startTimeInEpochSeconds}

@param applicationName:
Application name is the value that product may pass during each check out transaction. .
For more details to get the {application_name} please refer the user guide **("Feature Report Page (License Usage Pane")** table. Application name is case insensitive.

@param startTime:
Start time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time know as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

@param endTime:
End time of the duration for which the usage is to be retrieved. Time is represented as seconds using the standard base time know as "the epoch", for e.g January 1, 2016, 00:00:00 GMT is represented as 1451606400

Points to note:
By default, the response mediat type is of XML formation; however all GET APIs supports *JSON* format as well. To generate JSON the request's Accept header should have application/json.

Response

Response by default will be in the *XML format* which contains *HTTP* status code and usage for a given feature and duration.

Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	RESPONSE
200 OK	Below is the **success response** for the above query. Input value for {application_name} is LeanFT, {startTimeInEpochSeconds} is 1475605800 and {startTimeInEpochSeconds} is 1475692199.

```
<ApplicationUsage>
   <applicationName>LeanFT</applicationName>
   <usageStartTimeUTCInSeconds>1475605800</usageStartTimeUTCInSeconds>
   <usageEndTimeUTCInSeconds>1475692199</usageEndTimeUTCInSeconds>
   <productDetails>
      <productId>HP UFT</productId>
      <productVersion>12.52</productVersion>
      <pdfID>10027</pdfID>
      <pdfVersion>1.0</pdfVersion>
      <featureUsage>
         <feature>
```

```
            <featureId>10594</featureId>
            <featureVersion>1</featureVersion>
            <featureType>FLOATING</featureType>
            <featureDescription>HP Unified Functional Testing Concurrent User</fea
tureDescription>
         </feature>
         <capacityUsage>
            <peakUsage>5</peakUsage>
            <averageUsage>1</averageUsage>
         </capacityUsage>
         <checkoutsUsage>
            <totalCapacityCheckedout>10</totalCapacityCheckedout>
         </checkoutsUsage>
      </featureUsage>
   </productDetails>
   <status>SUCCESS</status>
</ApplicationUsage>
```

404 Not Found — **In case an invalid or unavailable application usage is queried**

```
<ApplicationUsage>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12005</errorCode>
      <errorMessage>Resource does not exist</errorMessage>
      <customMessage>Usage for resource SomeVale is not found.</customMessage>
      </errorDetail>
</ApplicationUsage>
```

200 OK — **In case of valid application name but usage in not available for the duration**

```
<ApplicationUsage>
   <status>FAIL</status>
   <errorDetail>
      <errorCode>12005</errorCode>
      <errorMessage>Resource does not exist</errorMessage>
      <customMessage>Usage for resource LeanFT is not found.</customMessage>
      </errorDetail>
</ApplicationUsage>
```

Points to note:

- Single application usage can be retrieved through one request.
- Application name is case insensitive.
- If client time out is set to 60 seconds and there are more than 1500 license transactions during the specified period, the response may timeout. Please ensure you increase the connection and read timout in this case

Licenses for Tokens

Retrieve Licenses associated with Tokens

Request

REQUEST

Method	URL
GET	https://<hostname>:<port>/autopass/wsservices/v9.6/token?floatingLicenseToken=20.67577AD..8D776

@**param** floatingLicenseToken:
Floating license token for LIVE or COMMUTER or REMOTE COMMUTER. The license token can be retrieved by caling the reportLicenseSet API in the client.

Response

Response by default will be in the *XML format* which contains *HTTP* status code and usage for a given feature and duration.

Also the status of this call can be identified through status field of the response object which will contain SUCCESS / FAIL.

STATUS	RESPONSE
200 OK	Below is the **success response** for the above query.


```
<LicenseListResponse>
   <licenses>
      <m_syncFlag>DEFAULT</m_syncFlag>
      <isDecrypted>false</isDecrypted>
      <m_licenseModel>0</m_licenseModel>
      <m_passwordType>1</m_passwordType>
      <m_featureID>23264</m_featureID>
      <m_encryptionType>4</m_encryptionType>
      <m_featureVersion>1</m_featureVersion>
      <m_productNumber>CSA-CODAR</m_productNumber>
      <m_ltu>1</m_ltu>
      <m_capacity>10</m_capacity>
      <m_extStartDate>1488326400</m_extStartDate>
      <m_extExpDate>900703</m_extExpDate>
      <m_ioDuration>0</m_ioDuration>
      <isIOStartFlag>false</isIOStartFlag>
      <m_ioStartDate>-1</m_ioStartDate>
      <m_productBundle>APSC</m_productBundle>
      <m_clusterInfo>asdfasd_padmavathir@hpe.com</m_clusterInfo>
      <m_annotation>HCM Express Edition</m_annotation>
      <m_signature>6d4ea903ad5b615412e65aab9c613d963b7c3515</m_signature>
      <m_createdTime>2283731822</m_createdTime>
      <m_actualCapacity>10</m_actualCapacity>
      <m_licenseID>6d4ea903ad5b615412e65aab9c613d963b7c3515</m_licenseID>
      <m_lockedCapacity>0</m_lockedCapacity>
      <m_deviceIDs>any</m_deviceIDs>
      <m_validfordays>-1</m_validfordays>
      <m_validformonths>-1</m_validformonths>
      <m_validfoyears>-1</m_validfoyears>
      <m_uom />
      <m_systemName />
      <m_purpose />
      <m_orderID />
```

```
      <m_SAID />
      <m_attibuteList />
      <m_gracePeriod>-1</m_gracePeriod>
      <m_graceCapacity>-1</m_graceCapacity>
      <m_gracePeriodType />
      <m_graceCapacityType />
      <m_subscriptionRenewalPeriod>-1</m_subscriptionRenewalPeriod>
      <m_subscriptionRenewalPeriodType />
      <m_productVersion />
      <m_licenseVersion />
      <m_rawLicense>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiIHN0YW5kYWxvbmU9InllcyI/Pgo8UmF3TGljZW5zZSB0eXBlPSJPdktleTQiPlFBREUgRDlFQSBIOVA5IEdIVVogVkhBNCBINjJWIFk5SkwgS01QTCBCODlIIE1aVlUgV1I0VyBMSFdFIDlNUkcgMzVSMiBDTVJHIEhQTVIgTUg1VSBBNTg5IDRGR0MgOVNZOSBSTEZWIExaQVEgUVJYRSBNOVZFIEtNVVQgVVlRUyBOWUtMIDRCUjQgRzJDRCA3RFQ5IE43MzUgTjhZRCBSS1BFIFlUUU4gTktZNiBLWUZTIFhQOFcgSktWVSBGSzgyIEY2QUQgVk1SQyBaQ1VGIERCM0ggM0JBQSBFOFY0IEI4VjYgNThLRSA1MkNVICJIQ00gRXhwcmVzcyBFZGl0aW9uIjwvUmF3TGljZW5zZT4=</m_rawLicense>
      <m_ioDaysRemaining>-1</m_ioDaysRemaining>
      <isValid>true</isValid>
      <isApscGenerated>false</isApscGenerated>
      <m_enablerLicense />
      <m_capacityLicense />
      <m_licenseFormat>OVKEY4</m_licenseFormat>
      <cryptoType>0</cryptoType>
   </licenses>
   <status>SUCCESS</status>
</LicenseListResponse>
```

Points to note:
By default, the response media type is of XML formation; however all GET APIs supports JSON format as well. To generate JSON the request's Accept header should have application/json.

A new API is introduced in AutopassJConcurrent class of AutoPass client to retrieve the details from the web service API exposed.

```
public List<License> getLicenseDetailSet(FloatingLicense floatingLicense) throws AutopassJException
```

JAVA Sample Code

Sample code to “GET” feature usage

```
URL url = new
URL("https://localhost:5814/autopass/wsservices/v9.3/usage/feature?featureId=10616&featureVersio
n=1&startTime=1475605800&endTime=1475692199");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setDoOutput(true);
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/xml");

//Authorization Header
String aplsUserName = "aplsUser";
String aplsPassword = "password";
BASE64Encoder enc = new sun.misc.BASE64Encoder();
String userpassword = new
StringBuffer(aplsUserName).append(":").append(aplsPassword).toString();
String encodedAuthorization = enc.encode(userpassword.getBytes());
StringBuffer encodeValue = new StringBuffer("Basic ").append(encodedAuthorization);
conn.setRequestProperty("Authorization", encodeValue.toString());


//Connect to Server
conn.connect();

if (conn.getResponseCode() != HttpURLConnection.HTTP_OK) {
       throw new RuntimeException("Failed : HTTP error code : "
              + conn.getResponseCode());
}

//Get response
BufferedReader br = new BufferedReader(new InputStreamReader(
       (conn.getInputStream())));

//Print response
String output;
System.out.println("Output from Server .... \n");
while ((output = br.readLine()) != null) {
   System.out.println(output);
}

//Disconnect Server
onn.disconnect();
```

Sample code to "POST/DELETE" (add/remove) members to/from Reservation Pool.

```
URL url = new URL("https://localhost:5814/autopass/wsservices/v9.3/reservation/pool/members");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setDoOutput(true);
conn.setRequestMethod("POST");//DELETE to remove members from the reservation pool
conn.setRequestProperty("Content-Type", "application/xml");
conn.setRequestProperty("Accept", "application/xml");

//Input xml
StringBuffer input = new StringBuffer("<ReservationPool>")
                          .append("<name>Development</name>")
                          .append("<members>")
                          .append("<userNameList>")
                          .append("<userName>ASIAPACIFIC/rpadmava</userName>")
                          .append("<userName>ASIAPACIFIC/ramana</userName>")
                          .append("</userNameList>")
                          .append("</members>")
                          .append("</ReservationPool>");

//Admin Authorization Header
String aplsUserName = "admin";
String aplsPassword = "password";
BASE64Encoder enc = new sun.misc.BASE64Encoder();
String userpassword = new
StringBuffer(aplsUserName).append(":").append(aplsPassword).toString();
String encodedAuthorization = enc.encode(userpassword.getBytes());
StringBuffer encodeValue = new StringBuffer("Basic ").append(encodedAuthorization);
conn.setRequestProperty("Authorization", encodeValue.toString());

OutputStream os = conn.getOutputStream();
os.write(input.toString().getBytes());
os.flush();

If not created
if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED || conn.getResponseCode() !=
HttpURLConnection.HTTP_OK) {
                    throw new RuntimeException("Failed : HTTP error code : "
                          + conn.getResponseCode());
}

BufferedReader br = new BufferedReader(new InputStreamReader(
                          (conn.getInputStream())));

String output;
System.out.println("Output from Server .... \n");
while ((output = br.readLine()) != null) {
             System.out.println(output);
}

conn.disconnect();
```

Points to note:

1. Use any Base64 encode options available to encode your username and password.
2. For “GET” APIs, “application/json” option is also available for header
3. HTTPS connection requires client side certificate verification. It is recommended to have a valid certificate at server and implement the certificate validation at client side. Refer “Using SSL Authentication in Java Clients” for Java implementation
4. Search for “Developing RESTful Web Service Clients”, to implement RESTful clients
5. Important point to note, if client time out is set to 60 seconds and there are more than 1500 license transactions during the specified period, the response may timeout. Please ensure you increase the connection and read timout in this case
6. The schema for the requests and responses are available at the APLS installer path <INSTALLED_PATH>\HP AutoPass License Server\HP AutoPass License Server\webapps\autopass\sdk\xsd

Errror Codes

ERROR CODE	ERROR INFO	CUSTOM ERROR INFO
12001	Pool Name does not exists	Pool by name ({pool_name}) doesn't exist at License Server
12002	Member already exist	• User by (x) already exists • IP by (x) already exists • Host by (x) already exists • Client ID by (x) already exists
12003	Member list is not defined	{userNameList} {ipAddressList} {hostList} {clientIDList} attributes are not defined
12004	IP address is not valid	Please enter valid IP address
12005	Member doesn't not exist	• User by **{value)** doesn't exist in **(pool1)** • IP by **{value)** doesn't exist in **(pool1)** • Host by **{value)** doesn't exist in **(pool1)** • ClientID by **{value)** doesn't exist in **({pool_name})**
12006	Failed to add member	• Empty message string is passed by User. Field value should not be null or empty • Empty message string is passed by IP. Field value should not be null or empty • Empty message string is passed by Host. Field value should not be null or empty • Empty message string is passed by Client Id. Field value should not be null or empty
12010	Invalid input	• Invalid value {value} for query parameter {@paramname}
12011	Authorization Failed	• User - {value}, does not have permission to perform this action
12012	Product not found	• Product {product_code} with version {product_version} is not configured in the License Server

Acronyms

APLS	AutoPass License Server
API	Application Program Interface
REST	Representational State Transfer