



VOSS



**VOSS Insights
Dashboard API Guide**

Release 23.3

Dec 13, 2023

Legal Information

- Copyright © 2023 VisionOSS Limited.
All rights reserved.
- This information is confidential. If received in error, it must be returned to VisionOSS ("VOSS"). Copyright in all documents originated by VOSS rests in VOSS. No portion may be reproduced by any process without prior written permission. VOSS does not guarantee that this document is technically correct or complete. VOSS accepts no liability for any loss (however caused) sustained as a result of any error or omission in the document.

DOCUMENT ID: 20231213174944

Contents

1	Introduction	1
1.1	Overview	1
2	Resources	2
2.1	assets	2
2.2	dashboards	8
2.3	system	8
3	Resources v2	10
3.1	Overview	10
3.2	/v2/login	10
3.3	/v2/lxt_updates	12
3.4	/v2/users	16
3.5	/v2/system	27
3.6	/v2/msteams	28
4	Appendix	29
4.1	References	29

1. Introduction

1.1. Overview

The API is broken up into the resources below. Each resource represents an object in the system. A resource will have associated data and a set of methods in which the user may operate on it. The following URIs currently do not require authentication.

Resource	Description
/assets	This resource will return data associated with all the assets discovered by the system. The Analytix will automatically discover assets if the asset is configured to send logs.
/dashboards	This resource can be used to create, update, and delete new dashboards.
/datasource	This resource can be used to create, update, and delete datasources.
/system	This resource will return data about the system in general.

2. Resources

2.1. assets

The assets resource supports the following operations.

Method	URL	Description
GET	/assets	Get a list of all assets.
GET	/assets/{asset_id}	Get a single asset by asset id.
POST	/assets	Add a new asset.
POST	/assets	Add multiple new assets.
PUT	/assets	Updates a single asset.
PUT	/assets	Updates multiple assets.
DELETE	/assets	Deletes a single asset.
DELETE	/assets	Deletes multiple assets.

2.1.1. GET

```
/assets  
/assets/{asset_id}
```

- Parameters
asset_id - Optional input parameter specifying the specific desired asset.

Example:

```
/assets/656
```

- Response Codes

HTTP Status Code	Reason
200	Success.

- Example Response

```
curl-kX GET https://10.13.37.12/api/assets/169
```

```
{
  "assets": [
    {
      "asset_id": "169",
      "asset_name": "10.13.37.1",
      "ipaddress": "10.13.37.1",
      "hostname": "10.13.37.1",
      "customer": "",
      "last_method": "snmp_trap",
      "last_byte_time": "1510782169"
    }
  ]
}
```

2.1.2. POST, PUT, DELETE Requests and Responses

Assets can be created, updated, and deleted using POST, PUT, and DELETE respectively. The same request parameter is required for all three requests. This section will describe the desired request parameter and resulting response output.

- Request Parameters

The `AssetInput` object describes the JSON object that can be used to modify a single Asset.

The `AssetInputArray` object describes the JSON object that can be used to modify multiple Assets.

Note: The names `AssetInput` and `AssetInputArray` will be referenced both in this documentation as well as the online interactive documentation that ships with the product.

- `AssetInput`

The following is an example of the required parameter for creating or modifying single Asset.

Note: The property `asset_id` is a required parameter for PUT (updates) and DELETE. It is optional and ignored for POST requests. The `asset_id` will be given as part of the response for a successful POST (insert) of an asset.

The API does not support the use of `ipaddress` as the property for updates and deletes. The property `ipaddress` is ambiguous because our system allows duplicate IP addresses to be inserted into the system. This gets even more complicated when we include the `customer` property in the discussion. In order to prevent updates and deletes to assets with duplicate IP addresses, this is a strict requirement for our API.

```
{
  "ipaddress":"(string) Ip address of asset.",
  "asset_id":"(long): Optional parameter on POST. Required parameter for PUT and
  ↪DELETE.",
  "hostname":"(string, optional): Hostname of the asset.",
  "did":"(string, optional): EM7 specific. Represents EM7 device id.", "mac_address":
  ↪"(string, optional): Mac address of the asset.",
  "address":"(string, optional): Physical address of the asset.",
  "model":"(string, optional): Model of device.",
  "version":"(string, optional): Version of device.",
  "manufacturer":"(string, optional): Manufacturer of device.",
  "timezone":"(string, optional): Timezone of device location.",
  "description":"(string, optional): Description of device.",
  "asset_name":"(string, optional): Name or alias of device.",
  "customer":"(string, optional): Customer name of device.",
  "site":"(string, optional): Site location name where device resides."
}
```

- AssetInputArray

The following is an example of the required POST parameter for adding multiple Assets.

Note: The property `asset_id` is a required parameter for PUT (updates) and DELETE. It is optional and ignored for POST requests. The `asset_id` will be given as part of the response for a successful POST (insert) of an asset.

```
[
  {
    "ipaddress":"(string) Ip address of asset.",
    "asset_id":"(long): Optional parameter on POST. Required parameter for PUT and
    ↪DELETE.",
    "hostname":"(string, optional): Hostname of the asset.",
    "did":"(string, optional): EM7 specific. Represents EM7 device id.",
    "mac_address":"(string, optional): Mac address of the asset.",
    "address":"(string, optional): Physical address of the asset.",
    "model":"(string, optional): Model of device.",
    "version":"(string, optional): Version of device.",
    "manufacturer":"(string, optional): Manufacturer of device.",
    "timezone":"(string, optional): Timezone of device location.",
    "description":"(string, optional): Description of device.",
    "asset_name":"(string, optional): Name or alias of device.",
    "customer":"(string, optional): Customer name of device.",
    "site":"(string, optional): Site location name where device resides."
  },
  {
    "ipaddress":"(string) Ip address of asset.",
```

(continues on next page)

(continued from previous page)

```

"asset_id": "(long): Optional parameter on POST. Required parameter for PUT and
↳DELETE.",
"hostname": "(string, optional): Hostname of the asset.",
"did": "(string, optional): EM7 specific. Represents EM7 device id.",
"mac_address": "(string, optional): Mac address of the asset.",
"address": "(string, optional): Physical address of the asset.",
"model": "(string, optional): Model of device.",
"version": "(string, optional): Version of device.",
"manufacturer": "(string, optional): Manufacturer of device.",
"timezone": "(string, optional): Timezone of device location.",
"description": "(string, optional): Description of device.",
"asset_name": "(string, optional): Name or alias of device.",
"customer": "(string, optional): Customer name of device.",
"site": "(string, optional): Site location name where device resides."
}
]

```

- Response Output
- Response Code

Note: Delete will only return a response code. 200

- Response Body

Below is an example of successful Insert or Update. Result may contain one or many assets.

The asset_id must be saved by the application in order to make any subsequent Updates or Delete requests.

AssetResult

```

{
  "assets":
  [
    {
      "asset_id": "656",
      "asset_name": "",
      "ipaddress": "192.168.1.1",
      "hostname": "test hostname",
      "customer": "Customer1",
      "last_method": "",
      "last_byte_time": "0"
    }
  ]
}

```

- Other Possible Responses

Response Code	Reason	Remedy
200	Success.	
400	Exception: UPDATE error: asset_id property required.	Include asset_id in request body
400	Exception: UPDATE error: Received null asset input.	See AssetInput
400	Exception: DELETE: error: asset_id property required.	Include asset_id in request body
400	Exception: DELETE error: Received null asset input.	See AssetInput

2.1.3. POST

/assets

- Example Curl Request

Command:

```
curl -k -w '\nRESP_CODE: %{response_code}\n'
-X POST https://10.13.37.12/api/assets
-d '{"did": "1234",
  "hostname": "test hostname",
  "ipaddress": "192.168.100.1",
  "alias": "alias natted ip address",
  "model": "test model",
  "version": "test version",
  "manufacturer": "test vendor",
  "timezone": "test timezone",
  "description": "test description",
  "address": "test physical address",
  "customer": "Test Customer1",
  "site": "Test Site1"}'
```

Output:

```
{"assets": [
  {"asset_id": "657",
   "asset_name": "",
   "ipaddress": "192.168.100.1",
   "hostname": "test hostname",
   "customer": "Test Customer1",
   "last_method": "",
   "last_byte_time": "0"}
]}
```

RESP_CODE: 200

2.1.4. PUT

/assets

- Example Curl Request

Note: asset_id was included.

Command:

```
curl -k -w'\nRESP_CODE: %{response_code}\n'  
-X PUT https://10.13.37.12/api/assets  
- d'{"asset_id": "657",  
    "did": "1234",  
    "hostname": "test hostname2",  
    "ipaddress": "192.168.100.1",  
    "alias": "alias natted ip address",  
    "model": "test model",  
    "version": "test version",  
    "manufacturer": "test vendor",  
    "timezone": "test timezone",  
    "description": "test description",  
    "address": "test physical address",  
    "customer": "Test Customer1",  
    "site": "Test Site1"}'
```

Output:

```
{"assets":  
  [{"asset_id": "657",  
    "asset_name": "",  
    "ipaddress": "192.168.100.1",  
    "hostname": "test hostname2",  
    "customer": "Test Customer1",  
    "last_method": "",  
    "last_byte_time": "0"}]}
```

RESP_CODE: 200

2.1.5. /assets - DELETE

/assets

Command:

```
curl -k -w'\nRESP_CODE: %{response_code}\n'  
-X DELETE https://10.13.37.12/api/assets  
- d'{"asset_id": "657",  
    "did": "1234",  
    "hostname": "test hostname2",  
    "ipaddress": "192.168.100.1",
```

(continues on next page)

(continued from previous page)

```
"alias":"alias natted ip address",
"model":"test model",
"version":"test version",
"manufacturer":"test vendor",
"timezone":"test timezone",
"description":"test description",
"address":"test physical address",
"customer":"Test Customer1",
"site":"Test Site1"}'
```

Output:

RESP_CODE: 200

2.2. dashboards

The dashboards resource supports the following operations.

Method	URL	Description
POST	/dashboards/data/user	Retrieves dashboard data for a specific user.
POST	/dashboards/user	Get a list of all available dashboards for a specific user.
PUT	/dashboards/user	Create or Update a dashboard for a specific user.
DELETE	/dashboards/user	Deletes a dashboard from a specific user.
POST	/dashboards/resources/ fields	Get a list of all available resources and their respective fields.
GET	/dashboards/roles	Get a dashboard role.
POST	/dashboards/roles	Add a dashboard role.
DELETE	/dashboards/roles	Deletes a dashboard role.

2.3. system

The system resource supports the following operations.

Method	URL	Description
GET	/system/stats	Get system stats.

- Example Output

Command:

```
curl -k -w '\nRESP_CODE: %{response_code}\n'
-X GET https://10.13.37.12/api/system/stats
```

Output:

```
{
  "data": {
    "Cpu - Idle": "74.48979591836735",
    "Cpu - Irq": "0",
    "Cpu - Nice": "0",
    "Cpu - Sys": "9.183673469387756",
    "Cpu - Total": "25.510204081632654",
    "Cpu - User": "16.3265306122449",
    "Load Percentage": "129.4189453125",
    "Load over Last 1 Minute": "2.58837890625",
    "Load over Last 5 Minute": "2.01416015625",
    "Load over Last 15 Minute": "1.71923828125",
    "Memory Free": 7368441856,
    "Memory Used": 9436925952,
    "Memory Total": 16805367808,
    "Memory Used Percent": "56.15423631196968",
    "Memory Free Percent": "43.84576368803032",
    "Number of Cores": 2,
    "Disk Used Percent": "17",
    "Disk Free Percent": "83",
    "Disk Total": 523089912,
    "Disk Used": 85928968,
    "Disk Free": 437160944,
    "customer": "VAADEMODASH",
    "hostname": "VAADEMODASH",
    "version": "sp62",
    "services": {
      "postgres": "running",
      "ndx_server": "running",
      "reporter": "running",
      "apache2": "running",
      "sshd": "running",
      "slapd": "running",
      "runit": "running"
    }
  }
}
```

RESP_CODE: 200

3. Resources v2

3.1. Overview

The following URIs will require authentication. The system currently implements a token based authentication system. Every resource under the v2 route requires a token property to be set in the http header. A token can be requested from the `/v2/login` URI.

Resource	Description
<code>/v2/login</code>	Use this resource to request a token.
<code>/v2/lxt_updates</code>	Use this resources to manage software updates for the product.
<code>/v2/users</code>	Use this resource to GET, POST, PUT, DELETE users.

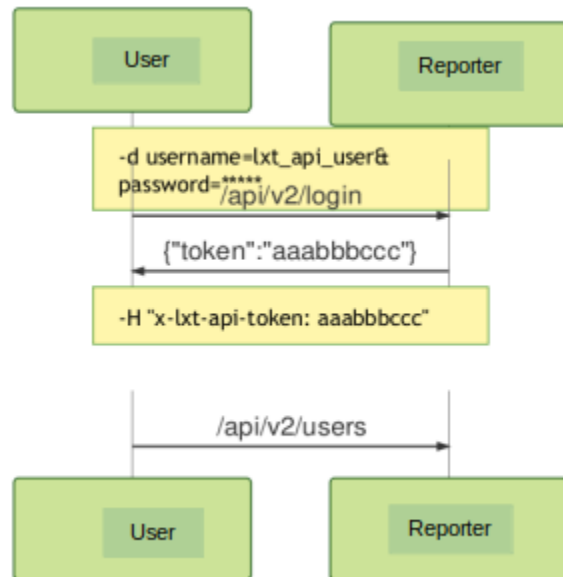
3.2. `/v2/login`

The `login` route is required before access to any route under v2 is requested. The system will respond with a token that needs to be included in the header of all subsequent API requests.

The following methods are supported for the `login` route.

Method	URI	Description
POST	<code>/v2/login</code>	Retrieves dashboard data for a specific user.

3.2.1. High Level login API Flow



3.2.2. POST

/v2/login

- Required Parameters

The login request requires a username and password parameter to be sent as part of the POST request.

Note: The username and password should be sent as a multipart form parameter. The username should be a userid that already exists in the system. A user can be added through our User Interface or via the API.

By default, the system contains a user named `lxt_api_user`. This userid can be used for first time API users. The `lxt_api_userpassword` is set at install time by your system administrator.

- Example Curl Request

Command:

```
curl -k -w '\nRESP_CODE: %{response_code}\n'
-X POST https://<IP or FQDN>/api/v2/login
-d "username=lxt_api_user&password=password1"
```

Output:

```
{"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]}"
```

RESP_CODE: 200

3.3. /v2/lxt_updates

Method	URL	Description
GET	/v2/lxt_updates	Retrieves current list of all update requests.
GET	/v2/lxt_updates/{id}	Retrieves information about a specific update request.
POST	/v2/lxt_updates	Adds a new update request.
PUT	/v2/lxt_updates	Modifies an existing update request.
PUT	/v2/lxt_updates/{id}	Modifies an existing update request.
DELETE	/v2/lxt_updates	Deletes an existing update request.
DELETE	/v2/lxt_updates/{id}	Deletes an existing update request.

3.3.1. Header (required)

x-lxt-api-token: "token from login"

3.3.2. GET

/v2/lxt_updates

- Example 1: Get All Updates

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/lxt_updates
```

Output (formatted):

```
::
{
  "status":200,
  "message":"Success",
  "data":[
    {"id":"12"},
    {"id":"13"}
  ]
}
```

```
RESP_CODE: ``200``
```

- Example 2: Get Updates with specified id

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/lxt_updates/12
```

Output (formatted):

```
{
  "status":200,
  "message":"Success",
  "data":[
    {
      "id":"12",
      "status":{"comment":"Error encountered. Please reference the install log."},
      "log":"This file does not look like a service pack.\n"
    }
  ]
}

RESP_CODE: ``200``
```

3.3.3. POST

Use POST to add a new update request.

/v2/users

- Input defines

The following definitions may be used when creating a new request. The new software should be copied into the drop account, or else provide a URL for fetching.

```
id=12 (optional)
delay=60 (optional)
url=http://www.layerxtech.com/downloads/arbitratorhawaii/updates (optional)
```

- Example 1: Add new update request

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X POST https://<IP or FQDN>/api/v2/lxt_updates
```

or


```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1Ni[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d 'id=12'
-d 'delay=60'
-d 'url=http://www.layerxtech.com/downloads/arbitratorhawaii/updates'
-X POST https://<IP or FQDN>/api/v2/lxt_updates
```

Output (formatted):

```
{
  "status":200,
  "message":"Success",
  "data":[{"id":"12"}]
}
```

RESP_CODE: 200

3.3.4. PUT

Use PUT to modify an existing update request. An error will be returned if the update request does not exist in the system.

/v2/lxt_updates

- Input defines

The following definitions may be used when creating a new request.

```
id=12 (optional)
delay=60 (optional)
url=http://www.layerxtech.com/downloads/arbitratorhawaii/updates (optional)
```

- Example 1: Update existing update request

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d 'id=12&delay=0'
-X PUT https://<IP or FQDN>/api/v2/lxt_updates
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d 'delay=0'
-X PUT https://<IP or FQDN>/api/v2/users/12
```

Output (formatted):

```
{
  "status":201,
  "message":"Success",
  "data":{"id":"12" }
}
```

RESP_CODE: 200

3.3.5. DELETE

/v2/lxt_updates

/v2/lxt_updates/{id}

- Example 1: Delete existing update request

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d"id=12"
-X DELETE https://<IP or FQDN>/api/v2/lxt_updates
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X DELETE https://<IP or FQDN>/api/v2/lxt_updates/12
```

Output (formatted):

```
{
  "status":201,
  "message":"Success",
  "data":[]
}
```

RESP_CODE: ``200``

- Example 2: Delete existing update request again

Command:

```
curl -s
-H x-lxt-apitoken:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
```

(continues on next page)

(continued from previous page)

```
-d"id=12"
-X DELETE https://<IP or FQDN>/api/v2/lxt_updates
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X DELETE https://<IP or FQDN>/api/v2/lxt_updates/12
```

Output (formatted):

```
{
  "status":404,
  "message":"Could not find existing entry for {id}",
  "data":[]
}
```

RESP_CODE: 404

3.4. /v2/users

Method	URL	Description
GET	/v2/users	Retrieves current list of all users.
POST	/v2/users	Adds a new user.
PUT	/v2/users	Modifies an existing user.
PUT	/v2/users/deactivate	Deactivate an existing user.
PUT	/v2/users/reactivate	Reactivate an existing user.
DELETE	/v2/users	Deletes an existing user.

3.4.1. Header (required)

x-lxt-api-token: "token from login"

3.4.2. GET

/v2/users

- Example 1: Get All Users

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/users
```

Output (formatted):

```
{
  "status":200,
  "message":"Success",
  "data":[
    {
      "email":"tset",
      "id":"IPM0N3CRFDHM4GQK15319464799176RQ299QFIJBA8B",
      "userId":"test",
      "firstName":"Bob",
      "lastName":"Smith",
      "status":"active",
      "customerId":null
    },
    {
      "email":"test",
      "id":"RUKTJDOYFSGJ4JO11532442929347PUPJAMFGP8TYE6",
      "userId":"loc",
      "firstName":"",
      "lastName":"",
      "status":"active",
      "customerId":null
    }
  ]
}
```

RESP_CODE: 200

- Example 2: Get Users with query parameter email

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/users?email=tset
```

Output (formatted):

```
{
  "status":200,
  "message":"Success",
  "data":[{"email":"tset",
    "id":"IPM0N3CRFDHM4GQK15319464799176RQ299QFIJBA8B",
    "userId":"test",
    "firstName":"Bob",
    "lastName":"Smith",
    "status":"active",
    "customerId":null
  }]
}
```

RESP_CODE: 200

- Example 3: Get Users with query parameter userId

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/users?userId=loc
```

Output (formatted):

```
{"status":200,
  "message":"Success",
  "data":[{"email":"test",
    "id":"RUKTJDOYFSGJ4J011532442929347PUPJAMFGP8TYE6",
    "userId":"loc",
    "firstName":"",
    "lastName":"",
    "status":"active",
    "customerId":null}]
}
```

RESP_CODE: 200

- Example 4: Get Users with an invalid email

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-X GET https://<IP or FQDN>/api/v2/users?email=invalid@email.com
```

Output (formatted):

```
{
  "status":400,
  "message":"User(s) not found.",
  "data":[]
}
```

RESP_CODE: 400

3.4.3. POST

Use POST to add a new user. An error will be returned if this user already exists in the system.

/v2/users

- Example Input JSON

The following JSON should be used when adding a new user to the system. POST parameters should be in JSON only.

v2users.add.user1

```
{
  "userId":"testuser1", (required)
  "password":"password1", (optional)
  "email":"testuser1@test.com", (required)
  "firstName":"TFirst", (optional)
  "lastName":"TLast", (optional)
  "customerId":"1234", (optional)
  "customerName":""," (optional, recommended if customerId is provided)
  "roles":[ (optional)
    {
      "name":"layerx_role1", (optional)
      "product":"Global SIP"(optional)
    }
  ]
}
```

- customerName

The API will automatically create a customer for the user if customerName is provided. The customerName will be returned with each subsequent API GET request. The customerName should be unique. New customers can be seen in the user interface under the **Access Controls -> Customer** area. The admin user can then add, modify, or delete resource filters to control what data this user can see.

- customerId

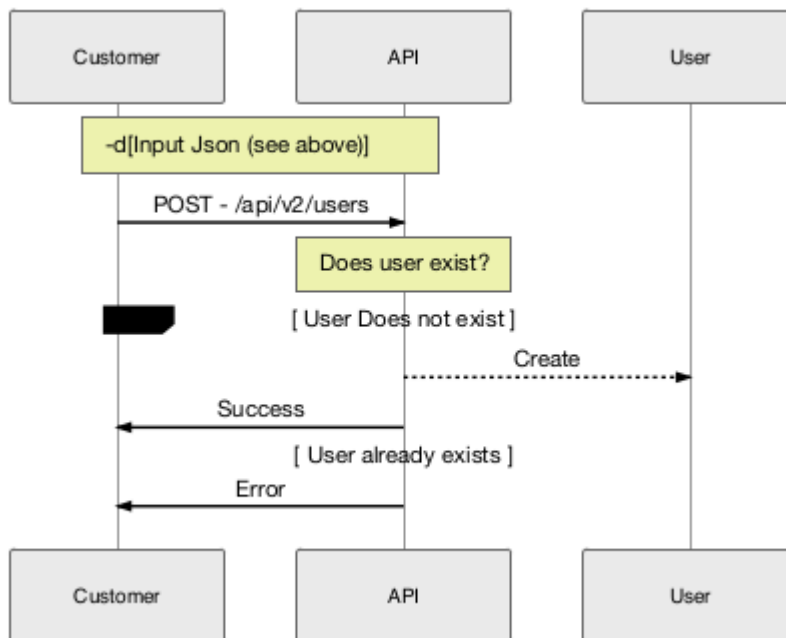
The API also supports the use of a `customerId`. The `customerId` is usually a customer specific unique identifier that has meaning to the end user. If `customerId` is provided, the API will use the `customerId` as the determining factor on whether or not to create a new customer. The API will automatically create a new customer for the user if `customerId` does not already exist in the system.

Note: `customerName` is still an optional field. The system still needs something user friendly to display to the end user in the user interface.

`customerId` is usually not very meaningful or useful to the end user. It is recommended that the `customerName` is also used in conjunction with `customerId` to provide a user friendly customer name.

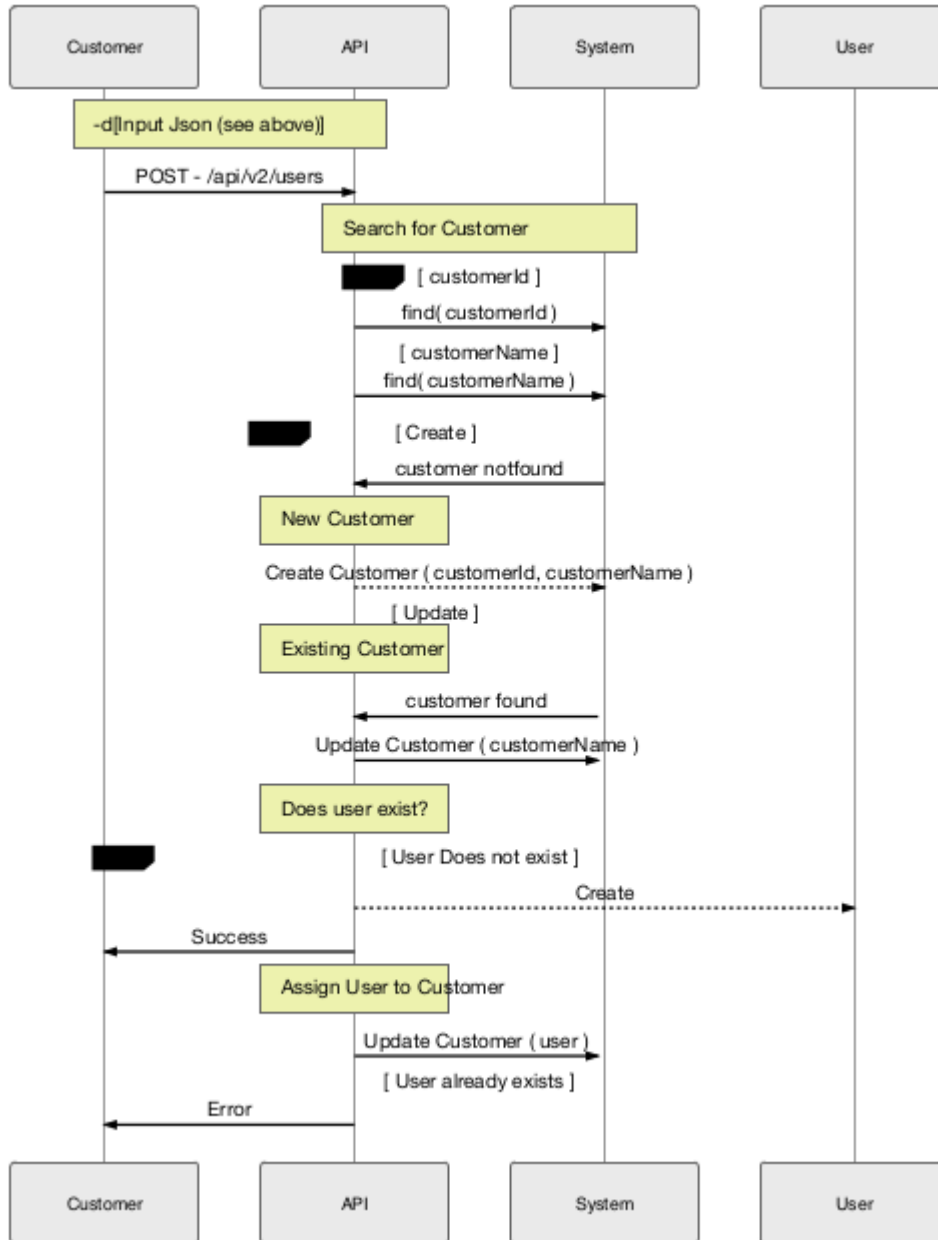
- POST High Level (No customer information)

The following flow demonstrates the system behavior when `customerId` and `customerName` are not provided.



- POST High Level (with customer information)

The following flow demonstrates the system behavior when `customerId` and `customerName` are provided.



- Example 1: Add new user

Command:

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[... ]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.add.user1
-X POST https://<IP or FQDN>/api/v2/users
  
```

or


```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"userId":"testuser1",
    "password":"password1",
    "email":"testuser1@test.com",
    "firstName":"TFisrt",
    "lastName":"TLast",
    "roles": [{"name":"layerx_role1",
               "product":"Global SIP"}]
    }'
-X POST https://<IP or FQDN>/api/v2/users

```

Output (formatted):

```

{"status":200,
 "message":"Success",
 "data":[{"email":"testuser1@test.com",
         "id":"4e37d6c336d2adbf52bbc5dda8[...]\"",
         "userId":"testuser1",
         "firstName":"TFirst",
         "lastName":"TLast",
         "status":"active",
         "customerId":null }]}

```

RESP_CODE: 200

- Example 2: Add an existing user

Command:

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.add.user1
-X POST https://<IP or FQDN>/api/v2/users

```

Output (formatted):

```

{"status":400,
 "message":"User already exists.",
 "data":[]}

```

RESP_CODE: 400

3.4.4. PUT

Use PUT to modify an existing user. An error will be returned if the user does not exist in the system.

```
/v2/users
/v2/users/deactivate
/v2/users/reactivate
```

- Example Input JSON

The following JSON should be used when modifying a new user to the system. PUT parameters should be in JSON only.

v2users.modify.user1

```
{
  "id": "xxxxxxxxxxxxxxxx", (optional)
  "userId": "testuser1", (required)
  "password": "password1", (optional)
  "email": "testuser1@test.com", (required)
  "firstName": "Firstname Changed", (optional)
  "lastName": "Last Name changed", (optional)
  "customerId": "1234", (optional)
  "customerName": "Customer B", (optional, recommended if customerId is provided)
  "roles": [ (optional)
    {
      "name": "layerx_role1", (optional)
      "product": "Global SIP" (optional)
    }
  ]
}
```

- customerName and customerId

Please refer to POST section for customerName and customerId parameters.

- Example 1: Update existing user

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.modify.user1
-X PUT https://<IP or FQDN>/api/v2/users
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"id": "xxxxxxxxxxxxxxxx",
  "userId": "testuser1",
  "password": "password1",
  "email": "testuser1@Changed",
```

(continues on next page)

(continued from previous page)

```

    "lastName":"Last Name changed",
    "customerId":"1234",
    "roles":[{"name":"layerx_role1",
              "product":"Global SIP"}]}'
-X PUT https://<IP or FQDN>/api/v2/users

```

Output (formatted):

```

{"status":201,
 "message":"Success",
 "data":{"email":"testuser1@test.com",
        "id":"4e37d6c336d2adbf52bbc5d[...]",
        "userId":"testuser1",
        "firstName":"Firstname Changed",
        "lastName":"Last Name changed",
        "status":"active",
        "customerId":"1234" }
}

```

RESP_CODE: 200

- Example 2: Deactivate user

Command:

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.modify.user1
-X PUT https://<IP or FQDN>/api/v2/users/deactivate

```

or

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"id":"xxxxxxxxxxxxxxxx",
    "userId":"testuser1",
    "password":"password1",
    "email":"testuser1@Changed",
    "lastName":"Last Name changed",
    "customerId":"1234",
    "roles":[{"name":"layerx_role1",
              "product":"Global SIP"}]}'
-X PUT https://<IP or FQDN>/api/v2/users/deactivate

```

Output (formatted):

```

{"status":201,
 "message":"Success",

```

(continues on next page)

(continued from previous page)

```

"data":{"email":"testuser1@test.com",
      "id":"4e37d6c336d2adbf52bb[...]",
      "userId":"testuser1",
      "firstName":"Firstname Changed",
      "lastName":"Last Name changed",
      "status":"inactive",
      "customerId":"1234"}
}

```

RESP_CODE: 200

- Example 2: Reactivate user

Command:

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.modify.user1
-X PUT https://<IP or FQDN>/api/v2/users/reactivate

```

or

```

curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"id":"xxxxxxxxxxxxxxxx",
    "userId":"testuser1",
    "password":"password1",
    "email":"testuser1@Changed",
    "lastName":"Last Name changed",
    "customerId":"1234",
    "roles":[{"name":"layerx_role1",
              "product":"Global SIP"}]}'
-X PUT https://<IP or FQDN>/api/v2/users/reactivate

```

Output (formatted):

```

{"status":201,
 "message":"Success",
 "data":{"email":"testuser1@test.com",
        "id":"4e37d6c336d2adbf52bbc[...]",
        "userId":"testuser1",
        "firstName":"Firstname Changed",
        "lastName":"Last Name changed",
        "status":"active",
        "customerId":"1234"}
}

```

RESP_CODE: 200

3.4.5. DELETE

Use DELETE to delete a user from the system.

/v2/users

- Example Input JSON

See POST and PUT section for example JSON. DELETE parameters should be in JSON only.

- Example 1: Delete existing user

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.add.user1
-X DELETE https://<IP or FQDN>/api/v2/users
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"id":"xxxxxxxxxxxxxxxx",
  "userId":"testuser1",
  "password":"password1",
  "email":"testuser1@Changed",
  "lastName":"Last Name changed",
  "customerId":"1234",
  "roles":[{"name":"layerx_role1",
    "product":"Global SIP"}]}'
-X DELETE https://<IP or FQDN>/api/v2/users
```

Output (formatted):

```
{"status":201,
 "message":"Success",
 "data":[]
}
```

RESP_CODE: 200

- Example 2: Delete existing user again

Command:

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d@test_data/v2users.add.user1
-X DELETE https://<IP or FQDN>/api/v2/users
```

or

```
curl -s
-H x-lxt-api-token:eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9[...]
--insecure
-w "RESP_CODE: %{response_code}"
-d '{"id":"xxxxxxxxxxxxxxxx",
    "userId":"testuser1",
    "password":"password1",
    "email":"testuser1@Changed",
    "lastName":"Last Name changed",
    "customerId":"1234",
    "roles":[{"name":"layerx_role1",
              "product":"Global SIP"}]}'
-X DELETE https://<IP or FQDN>/api/v2/users
```

Output (formatted):

```
{"status":404,
 "message":"Invalid user",
 "data":[]
}
```

RESP_CODE: 404

3.5. /v2/system

Method	URL	Description
GET	/v2/system	Retrieves current list of all update requests.
GET	/v2/system/daysRemaining	Retrieves days remaining on license.
GET	/v2/system/license	Retrieves license key.
PUT	/v2/system/productkey/{key}	Updates license key with {key}.

3.5.1. GET

/v2/system/daysRemaining

Output (formatted):

```
{"status": 200,
 "message": "Success",
 "data": {"daysRemaining": "171"}
}
```

/v2/system/license

Output (formatted):

```
{
  "status": 200,
  "message": "Success",
  "data": {"license": "9H3EJ-aaaaa-7X79K-nnnnn-cccc"}
}
```

3.6. /v2/msteams

Important: These APIs are strictly for the msteams collector. No other entity should need to use these routes.

The msteams collector can send JSON files directly to the Dashboard server for parsing.

Method	URL	Description
GET	/v2/msteams/tenant	Retrieves list of configured tenants
POST	/v2/msteams/tenant	Register a new msteams tenant

4. Appendix

4.1. References

4.1.1. Complete API Reference

Method	URL	Description
GET	/alerts	Get a list of all assets.
GET	/alerts/{alert_id}	Get a single asset by asset id.
GET	/assets	Get a list of all assets.

Method	URL	Description
GET	/assets/{asset_id}	Get a single asset by asset id.
POST	/assets	Add a new asset.
POST	/assets	Add multiple new assets.
PUT	/assets	Updates a single asset.
PUT	/assets	Updates multiple assets.
DELETE	/assets	Deletes a single asset.
DELETE	/assets	Deletes multiple assets.
GET	/system/stats	Get system stats.

4.1.2. /v2

Method	URL	Description
POST	/v2/login	Get API token
GET	/v2/lxt_updates	Get all update requests
POST	/v2/lxt_updates	Add update request
PUT	/v2/lxt_udpates	Modify update request
DELETE	/v2/lxt_udpates	Delete update request
GET	/v2/users	Get all users
POST	/v2/users	Add user
PUT	/v2/users	Modify user
PUT	/v2/users/deactivate	Modify user status to deactivated
PUT	/v2/users/reactivate	Modify user status to active
DELETE	/v2/users	Delete user
GET	/v2/msteams/tenant	Retrieves list of configured tenants. Strictly for the msteams collector.
POST	/v2/msteams/tenant	Register a new msteams tenant. Strictly for the msteams collector.

4.1.3. Status Codes

All status codes are standard HTTP status codes. The table below are status codes commonly used in the API.

Status Code	Description
200	OK
201	Created
202	Accepted
400	Bad Request by Client
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal LayerX Error
501	Not Implemented
503	Service Unavailable