



IxLoad

REST API Programming Guide

Version 8.40

Copyright and Disclaimer

Copyright © 2017 Ixia. All rights reserved.

This publication may not be copied, in whole or in part, without Ixia's consent.

Ixia, the Ixia logo, and all Ixia brand names and product names in this document are either trademarks or registered trademarks of Ixia in the United States and/or other countries. All other trademarks belong to their respective owners.

The information herein is furnished for informational use only, is subject to change by Ixia without notice, and should not be construed as a commitment by Ixia. Ixia assumes no responsibility or liability for any errors or inaccuracies contained in this publication.

RESTRICTED RIGHTS NOTICE

As prescribed by FAR 27.409(b)(4) and in accordance with FAR 52.227-14, please take notice of the following.

(a) This proprietary computer software and/or software technical data is submitted with restricted rights. It may not be used, reproduced, or disclosed by the Government except as provided in paragraph (b) of this notice or as otherwise expressly stated in the applicable contract.

(b) This computer software and/or software technical data may be—

(1) Used or copied for use with the computer(s) for which it was acquired, including use at any Government installation to which the computer(s) may be transferred;

(2) Used or copied for use with a backup computer if any computer for which it was acquired is inoperative;

(3) Reproduced for safekeeping (archives) or backup purposes;

(4) Modified, adapted, or combined with other computer software, provided that the modified, adapted, or combined portions of the derivative software incorporating any of the delivered, restricted computer software shall be subject to the same restricted rights;

(5) Disclosed to and reproduced for use by support service Contractors or their subcontractors in accordance with paragraphs (b)(1) through (4) of this notice; and

(6) Used or copied for use with a replacement computer.

(c) Notwithstanding the foregoing, if this computer software and/or software technical data is copyrighted computer software and/or software technical data, it is licensed to the Government with the minimum rights set forth in paragraph (b) of this notice.

(d) Any other rights or limitations regarding the use, duplication, or disclosure of this computer software and/or software technical data are to be expressly stated in, or incorporated in, the applicable contract.

(e) This notice shall be marked on any reproduction of this computer software, in whole or in part.

(End of notice)

Contacting Ixia

Corporate Headquarters	<p>Ixia Worldwide Headquarters 26601 W. Agoura Rd. Calabasas, CA 91302 USA +1 877 FOR IXIA (877 367 4942) +1 818 871 1800 (International) (FAX) +1 818 871 1805 sales@ixiacom.com</p>	<p>Web site: www.ixiacom.com General: info@ixiacom.com Investor Relations: ir@ixiacom.com Training: training@ixiacom.com Support: support@ixiacom.com +1 818 595 2599</p>
EMEA	<p>Ixia Europe Limited Part 2nd floor, Clarion House, Norreys Drive Maidenhead, UK SL6 4FL +44 (1628) 408750 FAX +44 (1628) 639916 salesemea@ixiacom.com</p>	<p>Support: support-emea@ixiacom.com +40 21 301 5699</p>
Asia Pacific	<p>Ixia Pte Ltd 210 Middle Road #08-01 IOI Plaza Singapore 188994</p>	<p>Support: support-asiapac@ixiacom.com +91 80 4939 6410</p>
Japan	<p>Ixia KK Nishi-Shinjuku Mitsui Bldg 11F 6-24-1, Nishi-Shinjuku, Shinjuku-ku Tokyo 160-0023 Japan</p>	<p>Support: support-japan@ixiacom.com +81 3 5326 1980</p>
India	<p>Ixia Technologies Pvt Ltd Tower 1, 7th Floor, UMIYA Business Bay Cessna Business Park Survey No. 10/1A, 10/2, 11 & 13/2 Outer Ring Road, Varthur Hobli Kadubeesanahalli Village Bangalore East Taluk Bangalore-560 037, Karnataka, India +91 80 42862600</p>	<p>Support: support-india@ixiacom.com +91 80 4939 6410</p>
China	<p>Ixia Technologies (Shanghai) Company Ltd Unit 3, 11th Floor, Raffles City, Beijing Beijing, 100007 P.R.C.</p>	<p>Support: support-china@ixiacom.com 400 898 0598 (Greater China Region) +86 10 5732 3932 (Hong Kong)</p>

This page intentionally left blank.

CONTENTS

Contacting Ixia	iii
New in this Release	vii
Before you Begin	ix
Chapter 1 REST Resources	1
Chapter 2 Supported Features	3
Chapter 3 Using the REST API over HTTPS	5
Chapter 4 REST Authentication	7
Chapter 5 Supporting Methods and Running Operations	11
Chapter 6 Operations	19
Chapter 7 IxLoadGateway: IxLoad Session Handling	29
Chapter 8 IxLoad Data Model	33
Chapter 9 Chassis Chain/Port Assignment Operations	35
Chapter 10 Statistics	45
Chapter 11 Logging	53
Chapter 12 REST Script Templates	55

This page intentionally left blank.

New in this Release

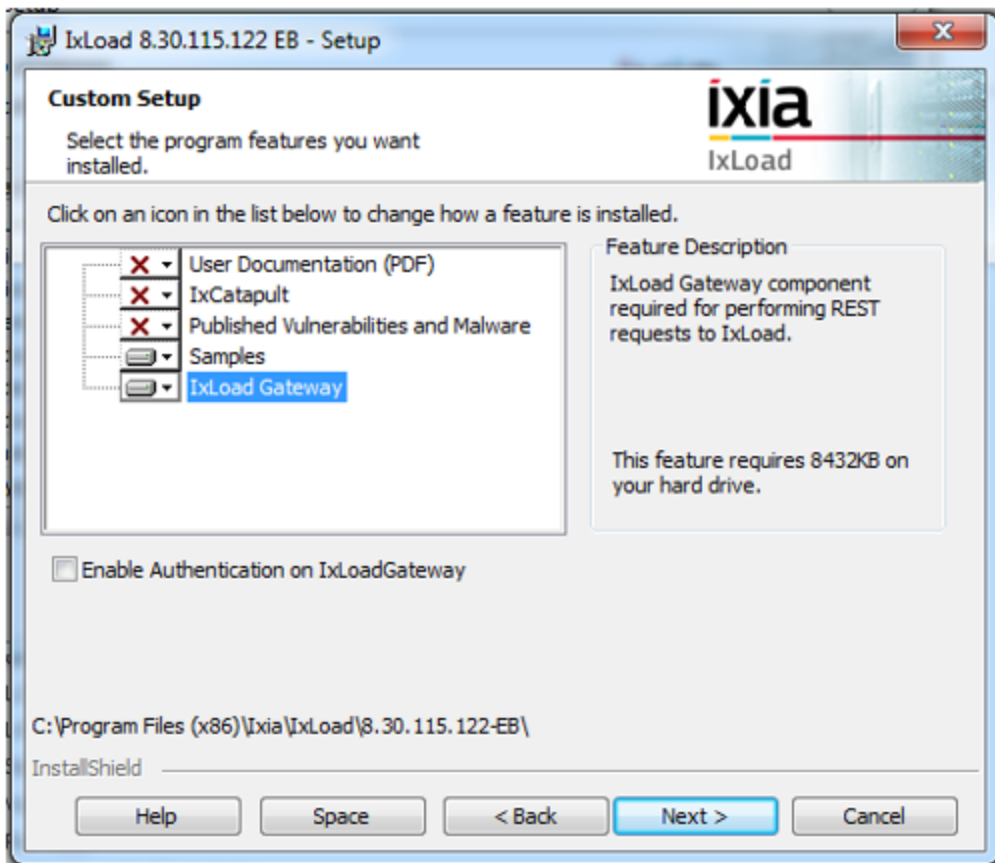
The following features are new in this release:

HTTP to HTTPS redirection	The REST API now redirects HTTP requests to HTTPS transport. See Using the REST API over HTTPS .
crf import	You can import crf files through the REST API.
waitForAllCaptureData	A request that waits for all the port data to be captured has been added.
Modifying the activity user objective on the fly	A request that changes the activity user objective while the test is running has been added. See Modifying the activity user objective value on the fly .
Activity filters	A request that adds an activity filter to a statistic has been added.

This page intentionally left blank.

Before you Begin

IxLoad REST API requires the IxLoad Gateway Service to be installed on the computer where you will use the REST API. The Gateway Service is an optional component that is not installed by default. To install it, click **Custom Setup** during IxLoad installation. If you have already installed IxLoad, click **IxLoad** from the list of installed applications (**Control Panel > Programs and Features**), right-click, and then click **Modify**. The installer runs and you can install the IxLoad Gateway Service.



This page intentionally left blank.

CHAPTER 1 REST Resources

A resource is a basic concept in terms of REST. This chapter defines the resource definition in terms of IxLoad.

A resource is a representation of an IxLoad object for the user. Not all IxLoad objects will be resources and not all IxLoad object functionality is exposed to the user.

A resource can have the following:

- Properties:
 - Primitives: Simple types like bool, int, and string.
 - Complex: Other resources like timeline resources and agent resources.
- Operations:
 - Example of operations:
 - RunTest
 - AddChassis
 - RefreshChassis

The IxLoad REST API allow you to start and configure an IxLoad session through REST API, through HTTP requests.

This page intentionally left blank.

CHAPTER 2 Supported Features

The following features are supported in the current release of the IxLoad REST API:

- Create and start an IxLoad session.
- Load a configuration (.rxf). The rxf is loaded from a local path.
- View data model tree, through GET requests (including query string support).
- Remove existing chassis and add new chassis.
- Assign and unassign ports.
- Change existing configuration and modify field values through PATCH requests.
- Support L23 range.
- Support L47 plugin.
- Save configuration modified through REST API.
- View, add, or delete the configured L23 and L47 statistics.
- Run test.
- Poll L23 and L47 statistics.
- Upload repository (.rxf) files.
- Start remote IxLoad sessions.
- Automatically generate documentation.
- Query logs from REST API.
- Analyze.

The following list of features are not supported in the current release of the IxLoad REST API:

- AppLibrary protocols/Resource Manager/Profiles (for example, real files)
- IxReporter
- Adding (POST)/removing (DELETE) objects such as test communities, plugins, and ranges. Add and remove operations are only supported officially on the chassis list, the port lists, and configured statistics.

This page intentionally left blank.

CHAPTER 3 Using the REST API over HTTPS

Starting with the IxLoad 8.40 release, requests made through IxLoad REST API are supported over both HTTP and HTTPS transport. The HTTP requests are redirected by IxLoadGateway to the HTTPS server and translated into HTTPS requests.

The default starting port for the IxLoadGateway HTTP server is 8080. Therefore, you can access IxLoadGateway through HTTP requests on a URL in the following format:

```
http://<IP_ADDRESS>:8080/api/v0/sessions
```

The default starting port for the IxLoadGateway HTTPS server is 8443. Therefore, you can access IxLoadGateway through HTTPS requests on a URL in the following format:

```
https://<IP_ADDRESS>:8443/api/v0/sessions
```

Self-signed certificates

HTTPS support over IxLoad REST API is offered through a self-signed certificate that is automatically generated by the IxLoad gateway component when it is installed as part of an IxLoad installation.

The self signed-certificate consists of two files:

- `ixload_certificate.crt`: This represents the actual self-signed certificate.
- `ixload_privkey.key`: This represents the private key used by the self-signed certificate.

Depending on the operating system on which the IxLoad build was installed, the self-signed certificate and its corresponding private key can be found at the following locations:

- On Windows, they can be found at `<IxLoadGateway_Install_Path>\certificate`.
- On the IxLoad Linux OVA, they can be found at `/opt/ixia/ixloadgateway/certificate`.

The self-signed certificate is generated by using a 2048 bit RSA keypair and the SHA-256 signature hash algorithm.

The self-signed certificate includes an X509 extension known as Subject Name Identifier (SNI)/ Subject Alternative Name (SAN). This extension allows the certificate to specify under which names (host names and IP addresses) a user can access a secured web server that is using that certificate. This prevents users from accessing IxLoad Gateway instances on different computers by using the same self-signed certificate.

For this extension, IxLoad gateway generates a log file named `san.log`, which contains all the host names and IPv4/IPv6 addresses under which the computer where IxLoad gateway is installed can be accessed. This log file resides in the same location as the auto generated certificate.

The certificate is regenerated automatically when one of the following occurs:

- The `ixload_certificate.crt`, `ixload_privkey.key`, or `san.log` files are deleted.
- The certificate has expired (it has a duration of 10 years).
- One of the entries required for SNI/SAN changes or disappears. For example, an IP address is changed, a host name is changed, or a network interface disappears.

Script changes required for HTTPS

The IxLoad REST scripts samples have been updated to support HTTPS requests over IxLoad REST API.

The changes are as follows:

- `kGatewayPort = 8443`: Changed from 8080 to 8443.
- `kResourcesUrl = 'https://%s:%s/api/v0/resources'`: Changed from `http` to `https`.

The utility files used by the IxLoad REST scripts samples have also been updated accordingly.

In `Utils\IxRestUtils.py`, the changes are as follows:

- `connectionUrl = "https://%s:%s/" % (server, port)`: Changed from `http` to `https`.
- `result = self._getHttpSession().request(method, absUrl, data=str(data), params=params, headers=headers, verify=False)`.

The `verify` parameter is a parameter provided by the `requests` library that is used in the REST scripts to generate HTTP/HTTPS requests. This parameter can take three values:

- `False`, as specified in the preceding example. If set to `False`, the HTTPS request does not perform any validation against a certificate.
- `True`, in this case, the HTTPS request performs a validation against a set of predefined certificate bundles specific to the Python `requests` module.
- `'<certificate_path>'`: In this case, the HTTP request performs a validation against the certificate specified at the path provided in the `verify` parameter.

To provide the certificate path, copy the certificate from the computer where the IxLoad gateway is installed to the computer where the REST script is run. The location where the certificate is copied is provided as the certificate path.

If the certificate is regenerated and the `verify` parameter is set to a certificate path in a REST script on a remote computer, that certificate will have to be downloaded again.

To run the IxLoad REST API sample scripts, the python executable needs to have the `pyOpenSSL` module installed.

Errors from REST UI clients

If you use a REST UI client such as Postman or advanced REST client, trying to access a URL from the IxLoad REST API might not work at first. This is because these two applications are tightly coupled to the Google Chrome browser. To be able to access any URL from the IxLoad REST API, you must first access one URL from the Google Chrome browser, accept the exception shown by the browser (because the web server uses a self-signed certificate), and then proceed to use the REST client.

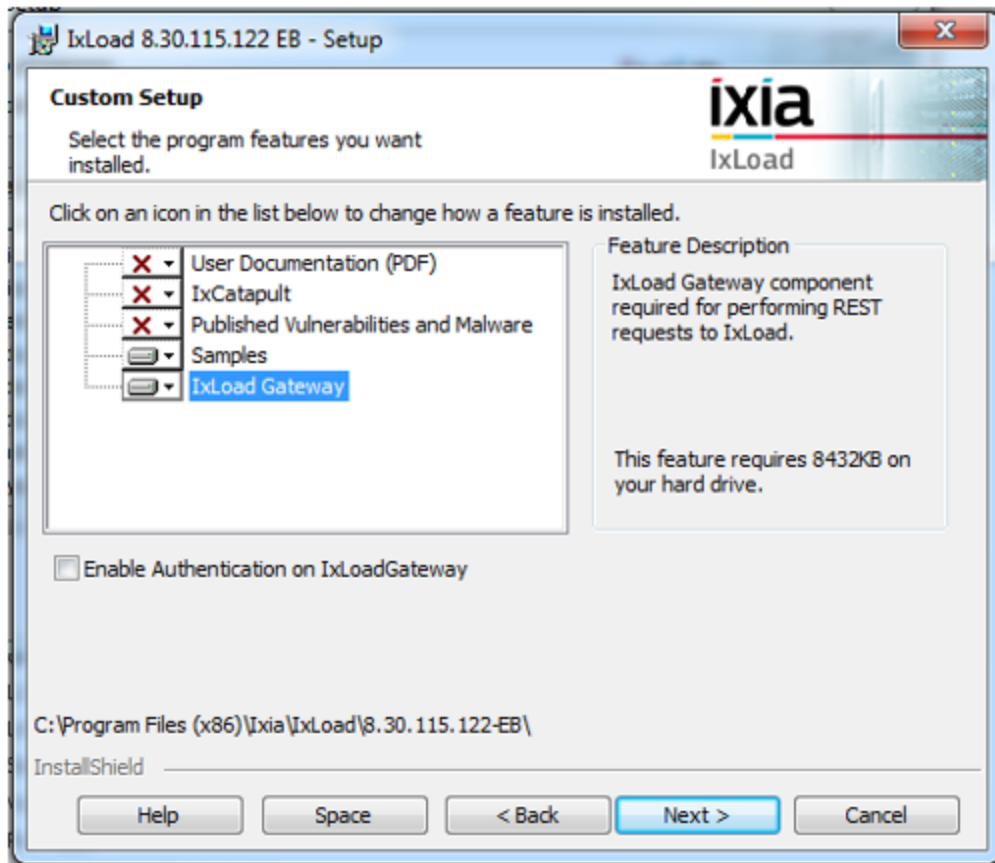
CHAPTER 4 REST Authentication

Starting with the IxLoad 8.30 Update 2 release, you can use the IxLoad REST API with authentication. After turning on authentication, most REST requests must include a unique user api-key. For the current release, this functionality is optional, integrated with Ixia User Management defined users only, and available for use with IxLoad versions running on Windows. REST Authentication is not available when you use IxLoad on Linux.

Prerequisites

Enabling authentication

To enable REST authentication, click **Custom Setup** during the IxLoad install process and choose the IxLoad Gateway feature. After selecting IxLoad Gateway, select the **Enable Authentication on IxLoadGateway** check box to turn on authentication.



The authentication feature can be turned on or turned off every time IxLoad and IxLoadGateway are installed. This means that if you install one IxLoad/IxLoadGateway version and turn on authentication, you install a newer version and if you do not select the **Enable Authentication on IxLoadGateway** check box, after the install is completed, authentication will be turned off.

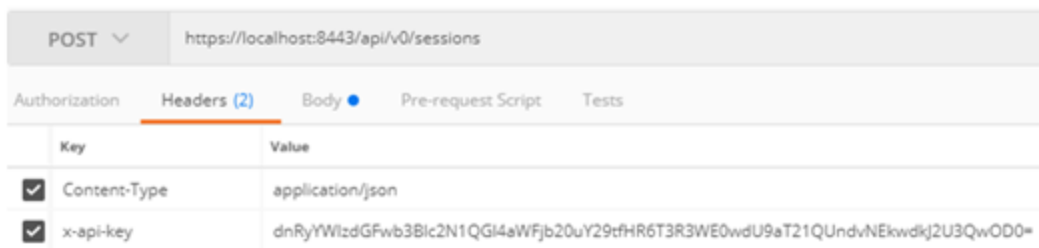
Ixia user management

Another prerequisite is that an instance of an Ixia user management server needs to be configured and present in the network, because for the current release, REST Authentication is only supported in conjunction with Ixia user management defined users.

User Management is a standalone application that you can download from the IxLoad section of Ixia's website (<https://support.ixiacom.com/support-overview/product-support/downloads-updates/versions/33>).

Authenticating REST requests

With 8.30 Update 2, an 'api-key' has been added to the request headers. This will be generated by the user management component based on a (username, password) pair. As a result, most requests will need to have an api-key present in their headers (see in the later section for the exceptions). An example can be seen in the following figure taken from the REST UI Client Solution Postman:



The only requests that are allowed without including an api-key in the requests are as follows:

- Getting the list of all created sessions: GET `https://localhost:8443/api/v0/sessions/`
- Getting the general status of a particular session: GET `https://localhost:8443/api/v0/sessions/1`

All other session-specific operations require the presence of an api-key.

After a session is created, the api-key provided is validated against the Ixia user management database through the user management server. If the key is not valid, an appropriate message is returned.

As part of all the other requests that manipulate a session, the api-key provided is compared with the api-key used to create that particular session.

The possible results when executing a request are as follows:

- If the operation was successful, a **201 Created** status or **200 OK** status is received.
- If the api-key was not specified in the headers, a **403 Forbidden** status is received, with the following message:
{

```
"status": "POST operation failed",
"error": "X-API-Key is not included in the header"
}
```

- If the api-key provided is not valid (does not exist in the user management database), a **403 Forbidden** status is received, with the following message:

```
{
"status": "POST operation failed",
"error": "The provided X-API-Key is not valid"
}
```

(This response is possible only for the CREATE session operation.)

- If the api-key is not valid for a session (not the same as the one that is used to create the session), a **403 Forbidden** status is received, with the following message:

```
{
"status": "POST operation failed",
"error": "X-API-Key mismatch"
}
```

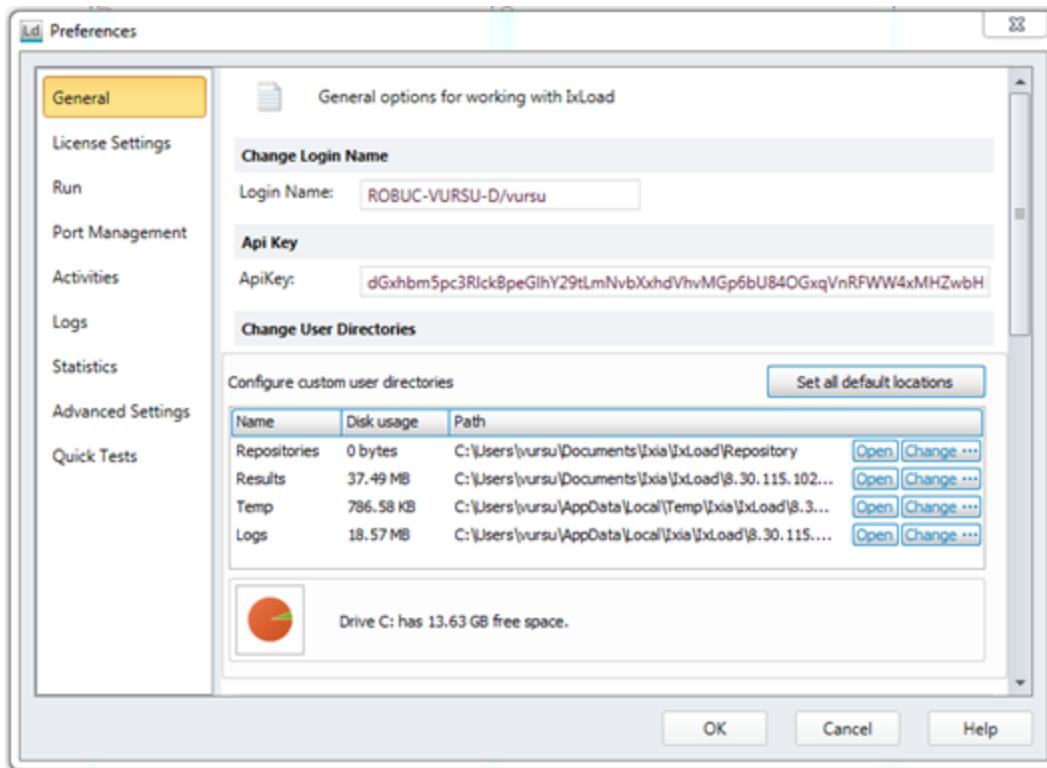
Users can delete only their own sessions (sessions that were created with the same api-key as the one provided during the DELETE request).

Retrieving the api-key

You can retrieve an api-key from the IxLoad UI.

In Ixload, when authentication is turned on and when you log on with your Ixia user management credentials, the api-key value can be retrieved from **File > Preferences > General**.

The value of the api-key automatically updates its value every time you change your password or when another user logs on. The field is read-only, so you can only copy the value of the api-key, but not modify it.



Script changes required for authentication

The changes that need to be made to IxLoad REST scripts for authentication are as follows:

- `kApiKey = ''`: If this value remains an empty string, the api-key will not be included in the header of the requests. Otherwise, it will be part of it.
- `connection.setApiKey(kApiKey)`: Setting the api-key for the connection.

CHAPTER 5 Supporting Methods and Running Operations

The IxLoad REST API is defined by how the resources are represented, by how they are accessed and changed, and by the exposed data model.

REST representation

The Ixload REST API handles a lot of different object types. Each object has amongst its values the following:

- Primitive values: These are basic values.
- Complex values: These will be represented by lists or other REST resources.

Primitive values

Primitive values (numbers, string, and bool) are used as values for REST options in the requests payload. These should be represented as follows:

- Strings are enclosed in quotes: "custom string," ""
- Numbers, integers, or float are not enclosed in quotes: 1, 1.1
- Booleans are not enclosed in quotes, and starts with lowercase: true, false

List objects

The IxLoad data model contains numerous lists. To be able to identify a resource that is part of a list (it must have a unique URL), the resource must have an ID associated, which is unique in the containing list. For this reason, each resource that is contained in a list has a field that contains its ID. This field is called 'objectID' in IxLoad. However, this name can be retrieved programatically by performing an 'OPTIONS' request on the resource, and retrieving the value for the 'resourceIdName' field. For now, this returns 'objectID.'

A resource's objectID can be retrieved by performing a GET request on the list, and iterating through the results, each element of the list (each resource) has this field set.

For a list that has the following URL:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList>

an element with objectID = 10 is retrieved by the following URL:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/10>

The requests listed in the following figures were made from the Google Chrome 'Advanced REST Client' add-on.

IxLoad REST API supports the following HTTP methods:

REST resources

Other REST resources are shown as links to another object. So each time an object is retrieved through the REST API, it may have primitive values, lists, and other REST objects. The other REST objects are shown as links that points to the data model location of the referenced REST object.

Case conventions

In IxLoad REST API, URLs are case insensitive. This applies to all parts of any URL, with the exception of the 'api' string at the beginning of the URL. This is not the case, however, for fields and values entered in request payloads. The field names entered in the payload are actually option names in IxLoad middleware, so the case defined must be followed.

Preferences

You can change several global options directly from the REST API by using the following URL:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/preferences>

The options that can be changed are shown in the following figure:

```
1 {
2   "continueTestOnLoadModuleFail": true,
3   "logCollectorSize": 100,
4   "links": [{}],
10  "maximumInstances": 3,
11  "enableDebugLogs": false,
12  "overloadProtection": true,
13  "autoRebootCrashedPorts": false,
14  "detailedChassisMonitoring": false,
15  "licenseModel": "Subscription Mode",
16  "checkLinkStateAtApplyConfig": true,
17  "ntpServer2": "10.215.170.83",
18  "ntpServer1": "0",
19  "csvThroughputUnits": "Bps",
20  "allowIPOverlapping": false,
21  "allowRouteConflicts": true,
22  "restObjectType": "ixRestPreferences",
23  "enableAnonymousUsageStatistics": false,
24  "licenseServer": ""
25 }
```

Note: IxLoad REST API sessions are started under the System user, not the user that you are logged on as. As all the global options except Maximum Instances, License Model, and License Server are saved per user, this means that settings made in the IxLoad UI has no effect on REST API runs, because the REST API is registered under the 'System' user. Therefore, for the Maximum Instances, License Model, and License Server options to have an effect on REST API tests, you must set them from the REST API.

These options can be changed by performing PATCH requests on the 'preferences' URL, with a payload as follows:

```
{"licenseServer":"ipOrHostname"}
```

IxLoad REST methods

The IxLoad REST API supports the following HTTP methods: GET, PATCH, POST, DELETE, and OPTIONS. The only content type we support for payloads is JSON. The payload applies to PATCH, POST, and DELETE methods.

GET

Users can make GET requests to receive the list of REST options for the requested resource. The GET request does not contain any payload. If the request is successful, a **200 OK** status is returned.

The result is a JSON dictionary containing the option names and values exposed by the resource. All the primitive options (bool, string, and int) are in the root dictionary, while complex options (other objects) are placed together, as a list, under the 'links' option. Each element of the 'links' list is a dictionary that contains the following:

- 'rel': The child resource name.
- 'href': A URL where the child resource can be accessed.

The following figure shows the output of a GET method applied to the activeTest REST resource in IxLoad:

The screenshot shows the Advanced REST Client interface. The URL bar contains `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest`. Below the URL bar, there are radio buttons for HTTP methods: GET (selected), POST, PUT, PATCH, DELETE, HEAD, OPTIONS, and Other. Below the methods, there are tabs for 'Raw', 'JSON', and 'Response', with 'JSON' selected. Under the 'JSON' tab, there are links for 'Copy to clipboard' and 'Save as file'. The main area displays the JSON response for the GET request:

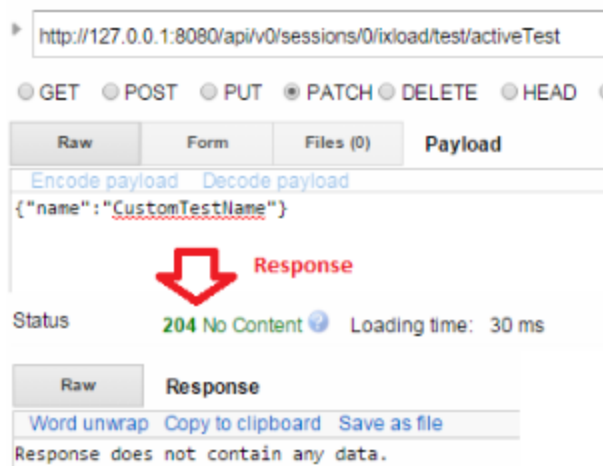
```
{
  comment: ""
  networkFailureThreshold: 0
  -links: [4]
    -0: {
      href: "/api/v0/sessions/0/ixload/test/activeTest/timelineList"
      rel: "timelineList"
    }
    -1: {
      href: "/api/v0/sessions/0/ixload/test/activeTest/totalUserObjectiveInfoList"
      rel: "totalUserObjectiveInfoList"
    }
    -2: {
      href: "/api/v0/sessions/0/ixload/test/activeTest/eventHandlerSettings"
      rel: "eventHandlerSettings"
    }
    -3: {
      href: "/api/v0/sessions/0/ixload/test/activeTest/captureViewOptions"
      rel: "captureViewOptions"
    }
  statViewThroughputUnits: "Kbps"
  showNetworkDiagnosticsFromApplyConfig: false
  csvThroughputScalingFactor: 1000
  activitiesGroupedByObjective: false
}
```

The preceding representation is only a demonstration of how the output of GET is provided in IxLoad REST API through the use of the Advanced REST Client in Google Chrome. The actual representation is different according to the programming language used to access the IxLoad REST API.

PATCH

You can perform PATCH requests to change field values on resources exposed by the IxLoad session. The PATCH request receives as payload a list of options that you wish to modify. Each pair in the dictionary contains a field name and the new option for it. If the request is successful, a **204 No Content** status is returned.

The payload for a PATCH request must contain at least one field that will be changed. This means one "field name": "new value" pair. The following figure shows the representation of the PATCH method in the advanced REST client:



Most resources cannot be modified by using PATCH requests while a test is running. If a PATCH request is made while a test is configuring or running, a **400 Bad Request** status is returned.

```
{
  status: "PATCH operation failed"
  error: "Cannot change HTTPClient1 at this moment. Please try again later"
}
```

POST

You can make POST requests to add elements to a list. The request is made on the list URL, and the actions that take place behind the scenes are to instantiate a new object of the type given by the list, and then to add the newly created object to the list. If the request is successful, a **201 Created** status is returned.

The payload for a POST request represents the parameters used when creating the resource that will be added to the list. Because not all resources (objects) require parameters in the constructor, the payload for a POST request can be empty (`{}`).

The following figure shows the output of the POST method in the advanced REST client:

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8080/api/v0/sessions`
- Method:** `POST` (selected)
- Headers:** (Empty)
- Payload:** `{"ixLoadVersion": "8.00.0.195"}`
- Content-Type:** `application/json`
- Status:** `201 Created` (loading time: 9 ms)
- Request headers:**
 - `User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36`
 - `Origin: chrome-extension://hgmlcoofddfdnphfgcellikdfbfjeloo`
 - `Content-Type: application/json`
 - `Accept: */*`
 - `Accept-Encoding: gzip, deflate`
 - `Accept-Language: en-US,en;q=0.8`
 - `Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464`
- Response headers:**
 - `Date: Fri, 13 Nov 2015 15:19:57 GMT`
 - `Content-Length: 2`
 - `Content-Type: application/json`
 - `Location: /api/v0/sessions/0`
 - `Server: CherryPy/3.6.0`

In the response headers, there is a field called **Location**, which contains the URL address of the newly created object.

Elements cannot be added to a list while a test is running. If a POST request is made while a test is configuring or running, a **400 Bad Request** status is returned.

```
{
  status: "POST operation failed"
  error: "Cannot perform the 'POST' operation at this moment. "
}
```

DELETE

You can make DELETE requests to delete one (or all) of the elements of the list. If the request is successful, a **204 No Content** status is returned.

DELETE requests do not require any payload.

If the DELETE request is made on a list URL, the list is cleared and all the elements are removed.

If the DELETE request is made on a URL that consists of the list URL and an object's unique ID appended at the end, only the object with that objectID is removed.

Example 1: DELETE on `http://127.0.0.1:8080/api/v0/sessions` deletes all sessions.

Example 2: DELETE on `http://127.0.0.1:8080/api/v0/sessions/2` deletes only the session with objectID = 2.

Elements cannot be removed from a list while a test is running. If a DELETE request is made while a test is configuring or running, a **400 Bad Request** status is returned.

```
{
  status: "DELETE operation failed"
  error: "Cannot perform the 'DELETE' operation at this moment."
}
```

OPTIONS

You can make OPTIONS requests on any resource. The output of the request is a set of information about the product and resource properties. If the result is successful, a **200 OK** status is returned.

OPTIONS requests do not require any payload.


As previously stated, in the OPTIONS response, there are two fields that specify the names of the unique object id field and the name under which all complex resources are kept on GET requests (the 'links' option name).

The following figure shows how the OPTIONS method output looks in the advanced REST client:

▶

GET
 POST
 PUT
 PATCH
 DELETE
 HEAD
 OPTIONS
 Other




Status **200 OK**  Loading time: 23 ms

Raw	JSON	Response
Copy to clipboard Save as file		
<pre> { -product: { version: "1.0.0.0" name: "eventhandlersettings" custom: null } -properties: [4] 0: "disabledEventClasses" 1: "disabledPorts" 2: "objectID" 3: "objectType" -features: { -rest: { multipost: false multidelete: true put: false patch: true typeName: "objectType" resourceIdName: "objectID" maxlist: null linksName: "links" } -session: { supported: true multiApp: true } -queryParam: { defaultEmbeddedValue: false embedded: false deepchild: false links: false includes: true } -auth: { authType: null } } } </pre>		

CHAPTER 6 Operations

Besides the HTTP requests listed earlier, executed on basic resources (objects or lists of the IxLoad data model), IxLoad REST API also offers support for operations. These are asynchronous actions performed on a certain resource (URL), that do not add, remove, or change field values for the resources that they are applied to, but rather change its state. Examples of operations are starting an inactive IxLoad session, connecting to an already existing chassis, or running a test.

To check when operations are available for a certain resource, perform a GET request on the resource URL, adding '/operations' at the end of the URL. The following figure shows how the operations available for the test REST resource are represented:



The screenshot shows a REST client interface. At the top, the URL `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/operations` is entered. Below the URL, there are radio buttons for HTTP methods: GET, POST, PUT, PATCH, DELETE, HEAD, and OPTIONS. The GET method is selected. A red arrow points to the 'JSON' tab in the response view. The response is displayed in a code editor with the following JSON structure:

```
{
  -loadTest: {
    fullPath: ""
  }
  runTest: {}
  -saveAs: {
    fullPath: ""
    overWrite: false
  }
  abortAndReleaseConfigWaitFinish: {}
  save: {}
}
```

The available operations are listed in the response, containing both the operation names and the parameters that they require. Default values for parameters are also shown.

Starting an operation

Starting an operation is done by performing a POST request on the following URL: `resourceUrl/operations/operationName`. The request payload represents the parameters required by

the operation, as shown in the preceding figure. Some operations (such as `runTest`) may not require any parameters, so for them, an empty payload must be sent: `{}`. The following figure shows the output of the `post` command for the `loadTest` operation:

The screenshot shows a REST client interface with the following details:

- URL: `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/operations/loadTest`
- Method: `POST` (selected)
- Request Headers:
 - User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2317.175 Safari/537.36
 - Origin: chrome-extension://hgmlfoofddfdnphfgcellkdfbfjeloo
 - Content-Type: application/json
 - Accept: *
 - Accept-Encoding: gzip, deflate
 - Accept-Language: ro-RO,ro;q=0.8,en-US;q=0.6,en;q=0.4
- Response Headers:
 - Date: Tue, 20 Oct 2015 21:00:12 GMT
 - Content-Length: 2
 - Content-Type: application/json
 - Location: `api/v0/sessions/0/ixload/test/operations/loadTest/1` (highlighted in red)
 - Server: Gunicorn/0.11.13
- Status: `202 Accepted` (Loading time: 18 ms)
- Payload: `{"fullPath": "stats.rxf"}` (highlighted in red)

uploadFile

```
uploadFile(connection, url, fileName, uploadPath, overwrite)
```

This operation uploads a file from the computer where the script runs, on the computer where the IxLoad client is running.

`connection` is the connection object that manages the HTTP data transfers between the client and the REST API.

`url` is the address of the resource that uploads the file. This `url` should be in the following form:

```
http://ip:port/api/v0/resources.
```

`filename` contains the name (or absolute path to the file, if the file is not in the same location as the executing script) of the file to be uploaded. This is the location on the computer where the script is running.

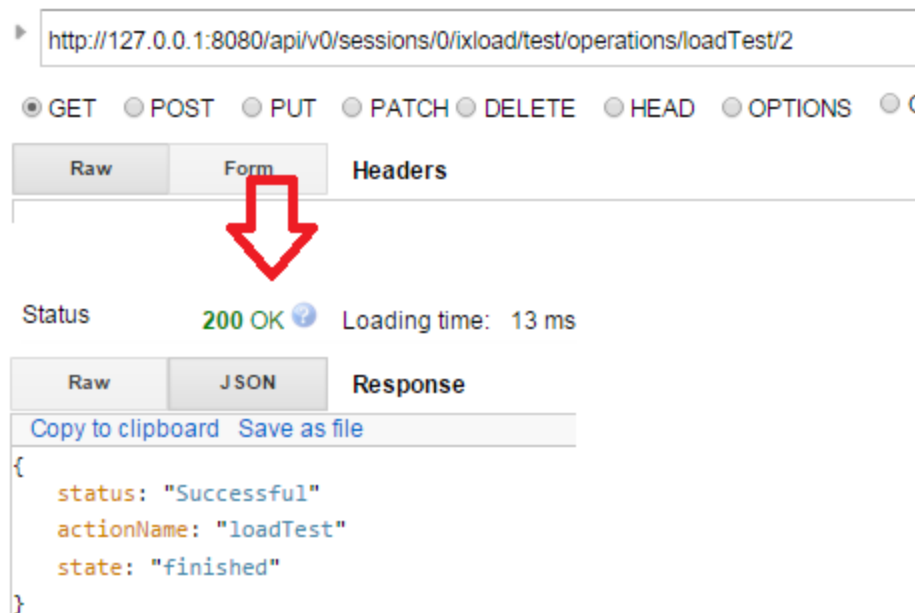
Example: `"file.txt", r"D:\\examples\\file.txt"`.

`uploadPath` is the path where the file should be copied to on the computer on which the IxLoad client runs.

`overwrite` specifies the required behavior if the file to be uploaded already exists on the remote computer. The default value is 'True.'

Getting an operation's status

Because these operations are asynchronous methods, you must be able to check an operation's status after starting it. For this, when starting an operation (executing the POST request), in the response header, there will be a field called **Location** that contains a URL. While performing a GET request on that URL, the operation's status will be returned. The following figure shows the output for getting the operation status belonging to the loadTest operation:



The screenshot shows a REST client interface with the following details:

- URL: `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/operations/loadTest/2`
- Method: GET (selected)
- Response Status: 200 OK (with a blue checkmark icon) Loading time: 13 ms
- Response Body (JSON):


```
{
  status: "Successful"
  actionName: "loadTest"
  state: "finished"
}
```

Possible values for the **state** field are as follows:

- Created: implies the operation was created.
- Executing: implies the operation is in progress.
- Finished: implies operation is done.

Possible values for the **status** field are as follows:

- Not started: implies the operation has not started yet, as operations are synchronous. It might be waiting for other operations to finish processing.
- In Progress: implies the operation is executed.
- Successful: implies the operations has finished and was successful.
- Error: implies the operation has finished but was not successful because an error has occurred.

In case the operation fails (exits with an error), in the preceding response, a new field is introduced, which contains the error message returned by the operation:

```
{
  status: "Error"
```

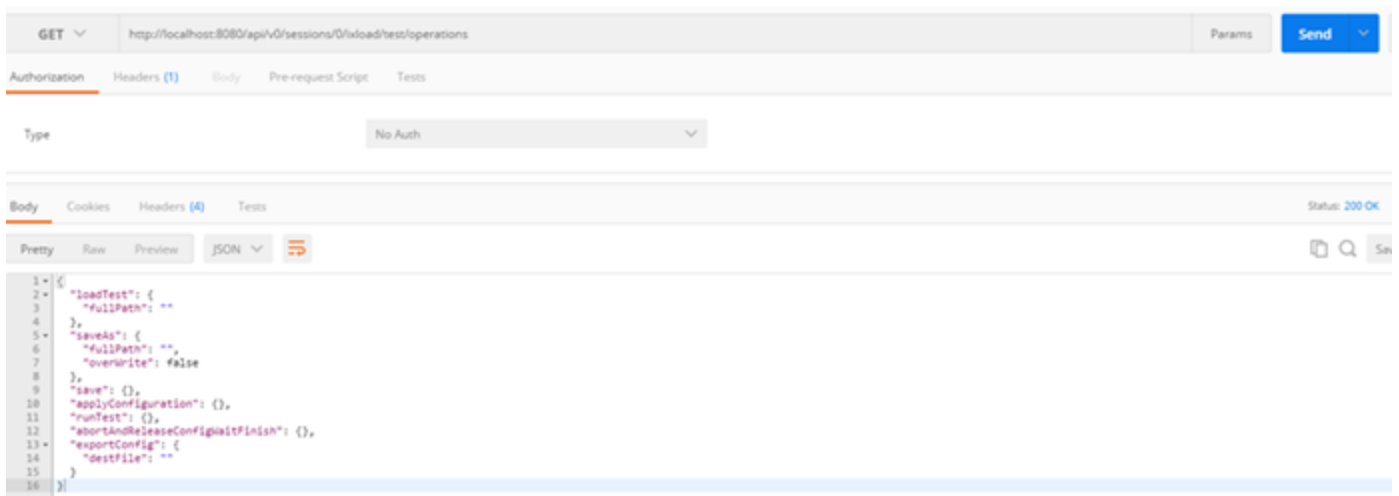
```
actionName: "loadTest"  
state: "finished"  
error: "File doesn't exist - F:\statsdfs.rxf"  
}
```

! **Important!** The URL where an operation's status can be retrieved is only active for a certain amount of time. At the moment, this lifetime is of 10 minutes. If a GET request on the operation URL/operationID URL is performed after this period, the status is not retrieved, but a '400 Bad Request' error is returned.

Examples of common operations in the IxLoad REST API

A list of the most commonly used operations for an IxLoad test in the REST API can be obtained by doing a GET on <http://localhost:8080/api/v0/sessions/0/ixload/test/operations>. The result lists the following operations (also shown in the figure below):

- **loadTest:** Load an IxLoad configuration file. The fullPath of the rxf to be loaded will need to be passed on as a parameter.
- **importConfig:** Import a .crf file as the current test configuration. The location of the .crf file and the location where the .rxf file will be saved after the import need to be passed as parameters.
- **saveAs:** Save the currently-loaded configuration file as a new file. The new file path for the rxf will need to be passed as a parameter, and the overwrite option in case the file path already exists.
- **save:** Save the currently loaded configuration file.
- **applyConfiguration:** Apply configuration on the current IxLoad test. The test will go to the Configured state. This is equivalent to clicking **Apply Config** in the IxLoad UI.
- **runTest:** Run the current IxLoad test. The test will go to the running state directly. This action is equivalent to clicking **Run test** in the IxLoad UI.
- **waitForAllCaptureData:** Wait for the test to capture all the port data that was received after the test has finished running.
- **abortAndReleaseConfigWaitFinish:** Stop the currently running IxLoad test.
- **exportConfig:** Export the currently loaded configuration file as a .crf file. The location of the archive needs to be passed as a parameter.



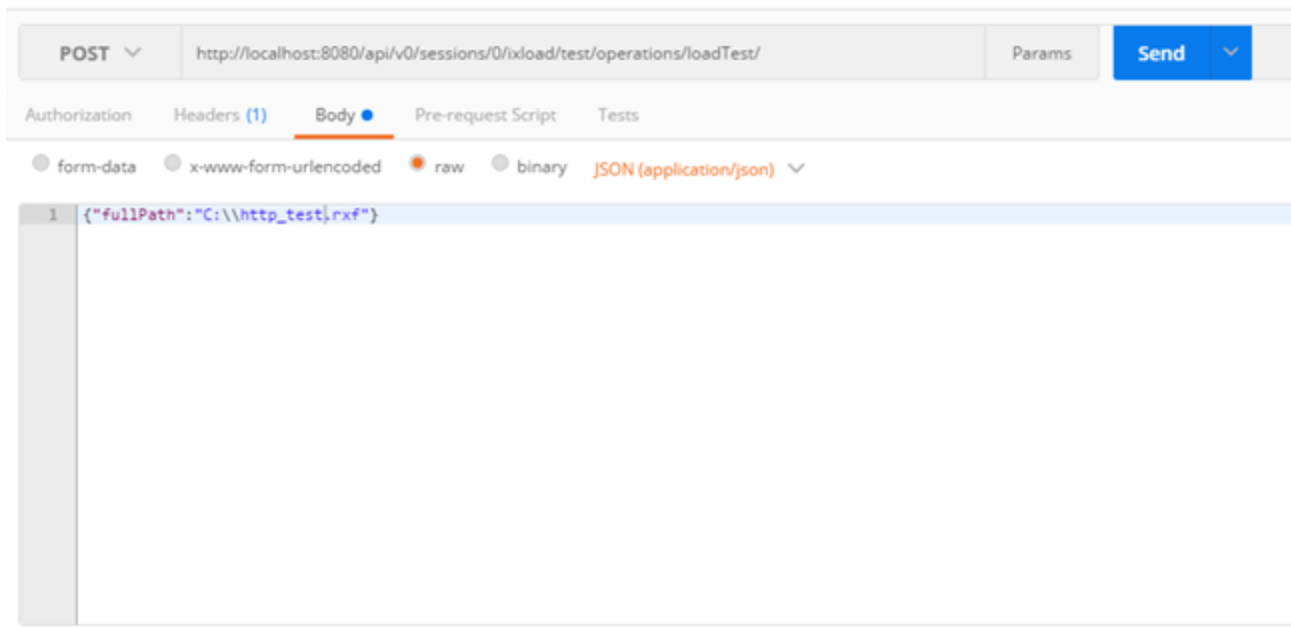
Example for loading a repository (.rxf) file

On an already created and activated session, do a POST on a URL similar to the following:

`http://localhost:8080/api/v0/sessions/[SESSIONID]/ixload/test/operations/loadTest/`

In the payload or the body of the request, add the path to the .rxf file.

```
{"fullPath": "C:\\http_test.rxf"}
```



As described in [Getting an Operation's Status](#), query the status of the operation until the state is **Finished**.

Example for importing a .crf file

On an already created and activated session, do a POST on an URL similar to the following:

```
http://localhost:8080/api/v0/sessions/  
[SESSIONID]/ixload/test/operations/importConfig
```

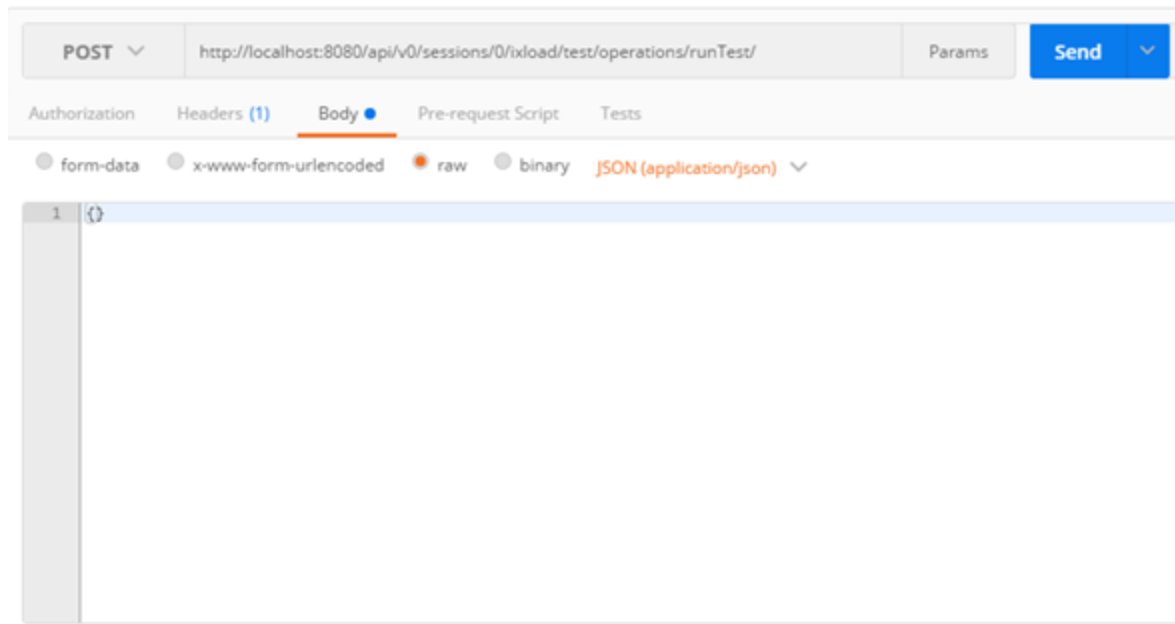
In the payload and body of the request, add the path to the .rxf file:

```
{"srcFile": "C:\\mycrf.crf", "destRxf": "C:\\rxf_from_crf.rxf"}
```

Example of running a test

On an already created and activated session in which there is either a loaded configuration file or a new test has been created, do a POST on a URL similar to the following:

```
http://localhost:8080/api/v0/sessions/[SESSIONID]/ixload/test/operations/runTest/
```



As described in [Getting an Operation's Status](#), query the status of the operation until the state is **Finished**.

Example for waiting to capture the port data

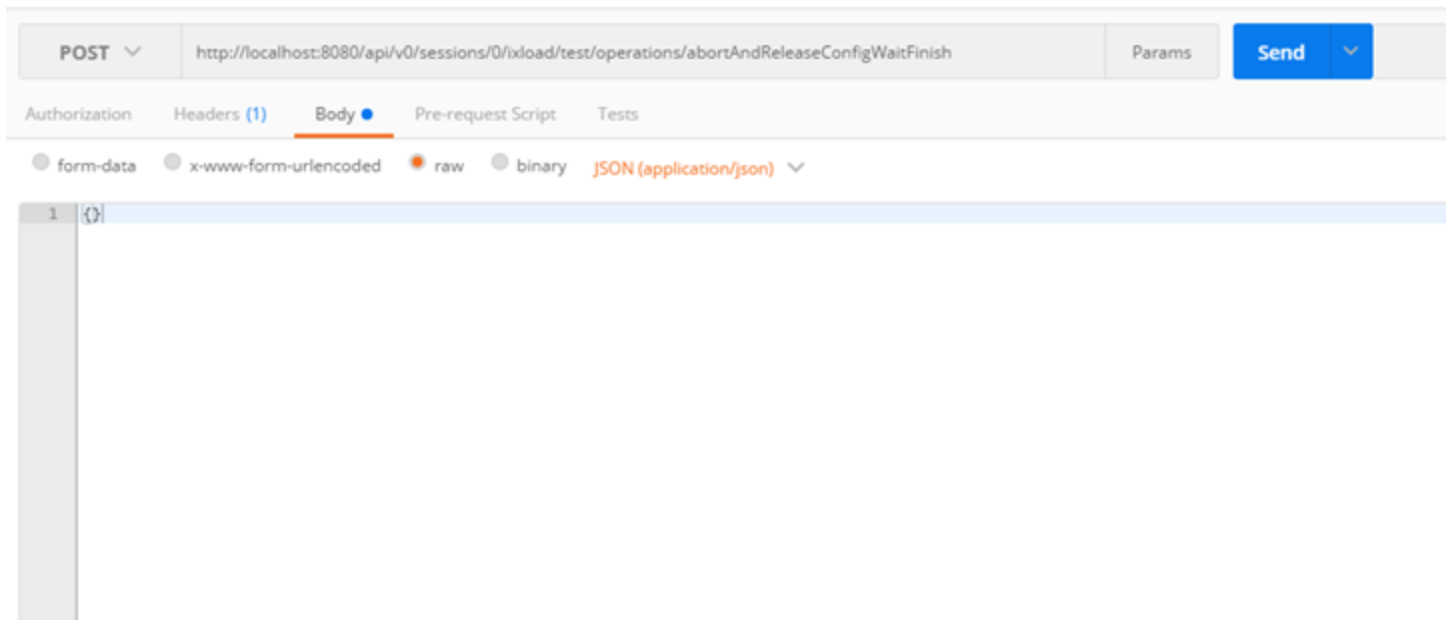
On an already created and activated session, do a POST on an URL similar to the following:

```
http://localhost:8080/api/v0/sessions/  
[SESSIONID]/ixload/test/operations/waitForAllCaptureData
```

Example of stopping a test

On an already created and activated session in which there is either a loaded configuration file or a new test has been created, do a POST on a URL similar to the following:

```
http://localhost:8080/api/v0/sessions/  
[SESSIONID]/ixload/test/operations/abortAndReleaseConfigWaitFinish
```



As described in [Getting an Operation's Status](#), query the status of the operation until the state is **Finished**.

Query strings

You can search by using a filter with one or more parameters separated by commas.

Note: The format for query strings changed between release 8.00 and 8.10.

For 8.00, the format was as follows:

```
http://resourceUrl?fieldName=value
```

Beginning with 8.10, the format is as follows:

```
http://resourceUrl?filter="fieldName <operator> value
```

The query strings are inserted under the 'filter' parameter, at the end of the URL. The supported query string operators are as follows:

- eq: equals
- ne: not equal to
- lt: lower than
- gt: greater than

- le: lower or equal to
- ge: greater or equal to

When the 'eq' operator is used for string fields (for example, names of statistics), it automatically has a 'contains' effect. This means that a GET request on `/configuredStats?filter="caption eq HTTP"` returns all statistics whose caption contains "HTTP." If instead, a 'matches' operation is required, you can still use 'eq,' but the value must be enclosed in quote marks (""). This causes a GET on `/configuredStats?filter="caption eq \"HTTP\""` to return only those statistics whose caption is exactly "HTTP."

Multiple query string conditions can be introduced in the same URL, separated by commas.

For example, the following URL returns all enabled statistics whose objectID is less than or equal to 14:

- `http://localhost:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats?filter="enabled eq True,objectID le 14"`

Query Strings are only supported on list resources, with the following methods:

- GET: Returns all the elements of the list, which satisfy the query string conditions.
- PATCH: Modifies the parameter list sent in the request payload with all the elements of the list, which satisfy the query string conditions.
- DELETE: Deletes from the list all the elements of the list, which satisfy the query string conditions.

Collecting diagnostics

IxLoad includes a diagnostics collection utility that collects log files and packages them into a ZIP file, so that they can be stored or they can be sent over an email conveniently. In the GUI, access the utility from **File > Tools > Diagnostics**. You can collect those same log files by using the REST API.

To collect diagnostics, ensure the following:

- At least one session must be active.
- The test must be in either the Configured or Unconfigured state.

To collect diagnostics, use the following command:

```
POST @ api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics
```

Specify the ZIP file location as the POST payload:

```
{"zipFileLocation": "<path to save ZIP file>"}
```

For example:

```
{"zipFileLocation": "C:\\Users\\ixia\\Desktop\\diags.zip"}
```

The following figure shows an example of a POST operation to collect diagnostics from a REST client:

> <http://127.0.0.1:8080/api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics>

GET
 POST
 PUT
 DELETE
 Other methods
 application/json

Raw headers Headers form Headers sets

```
Content-Type: application/json
```

Raw payload Data form Files (0)

```
{"zipFileLocation": "C:\\Users\\ixia\\Desktop\\diags.zip"}
```

SEND

Status: **202: Accepted** ? Loading time: 27 ms

Response headers (5) Request headers (2) Redirects (0) Timings

```
Date: Mon, 01 Aug 2016 08:50:50 GMT
Content-Length: 2
Content-Type: application/json
Location: api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics/1
Server: CherryPy/3.6.0
```

Raw JSON

```
{}
```

The status of the POST operation to collect diagnostics should be **202:Accepted**. The response to the operation should include a location.

To query the status of the POST operation, use a GET operation and specify the location received in the response to the POST.

For example:

```
GET @
http://127.0.0.1:8080/api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics/1
```

The following figure shows an example of a query to get the status of a diagnostics collection operation:

> <http://127.0.0.1:8080/api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics/3> ⋮

GET POST PUT DELETE Other methods ▾

Raw headers Headers form Headers sets

Content-Type: application/json

SEND

Status: 200: OK ? Loading time: 25 ms

Response headers (4) Request headers (1) Redirects (0) Timings

Date: Mon, 01 Aug 2016 09:03:59 GMT
Content-Length: 118
Content-Type: application/json
Server: CherryPy/3.6.0

Raw JSON

```
{
  "status": "In Progress"
  "actionName": "collectDiagnostics"
  "state": "executing"
  "result": ""
}
```

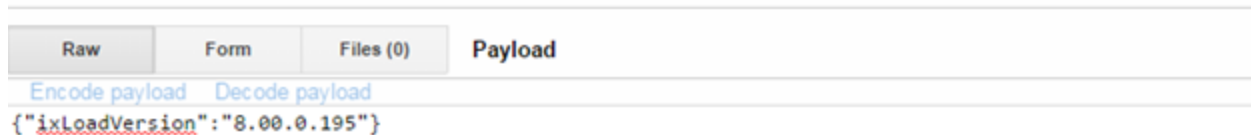
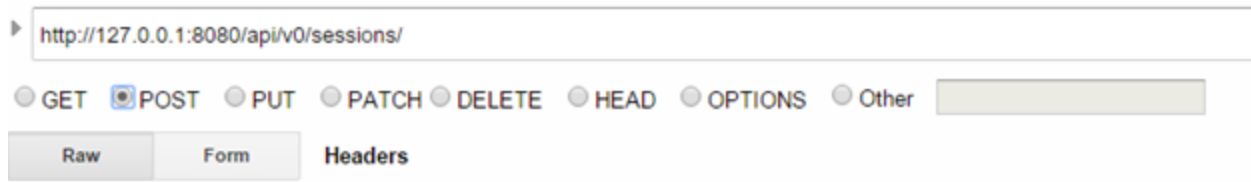
CHAPTER 7 IxLoadGateway: IxLoad Session Handling

Creating and handling IxLoad sessions is done through an IxLoad service, named IxLoadGateway. IxLoadGateway is installed with IxLoad as part of the custom install options.








Creating a new session

To create a new session object, a POST on `api/v0/sessions` with payload: `{"ixLoadVersion": "version no."}` needs to be performed. Note that this means the session is just created, but not started and not active.

This will not take into consideration the instance count limit on the client side. This will only work for sessions that are started. The following figure shows how this looks from rest client:

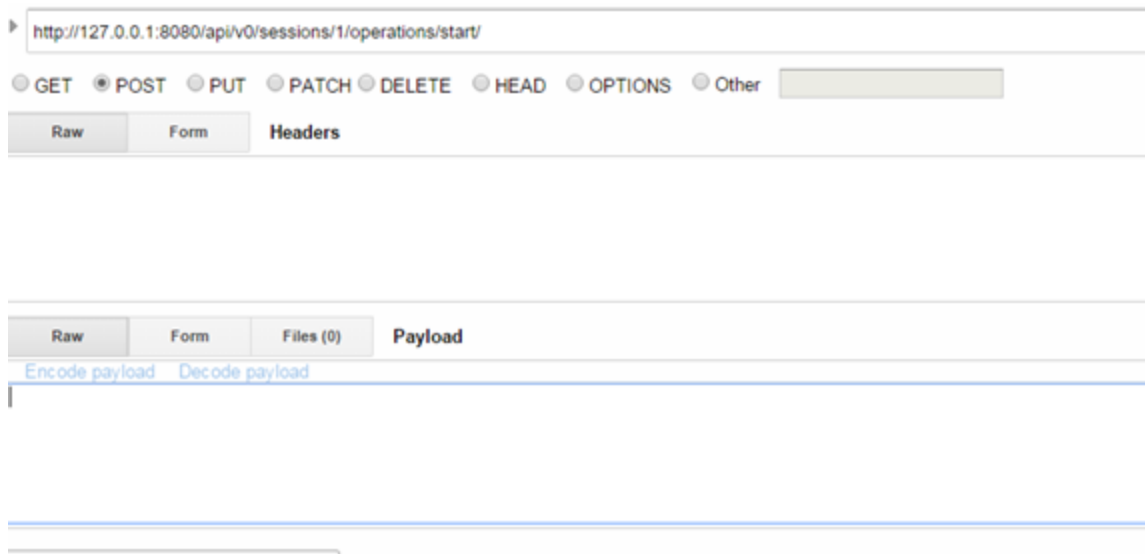


The following figure shows the response for the preceding post. Status is **201 Created** and the location points to the session.

Status	201 Created  Loading time: 7 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Origin: chrome-extension://hgmloofddfdnphfgcellkdfbfjello Content-Type: application/json  Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.8 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
Response headers	Date: Wed, 07 Oct 2015 14:36:15 GMT  Content-Length: 2  Content-Type: application/json  Location: /api/v0/sessions/1  Server: CherryPy/3.6.0 

Starting a session

To start a session, an operation is provided, named start. This operation is available on each individual session and requires no payload. The following figure shows how this looks in the REST client. This operation starts a new IxLoad session based on the IxLoad version for which the session was created.



The following figure shows the response for the start operation. Response is **202 Accepted**. The location shows the result for the operation.

Status 202 Accepted Loading time: 14 ms

Request headers
 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
 Origin: chrome-extension://hgml0o0dffdnphtgcellkdfbfbjeloo
 Content-Type: application/json
 Accept: */*
 Accept-Encoding: gzip, deflate
 Accept-Language: en-US,en;q=0.8
 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464

Response headers
 Date: Wed, 07 Oct 2015 14:43:41 GMT
 Content-Length: 2
 Content-Type: application/json
 Location: api/v0/sessions/1/operations/start/1
 Server: CherryPy/3.6.0

The following figure shows how the operation result for start looks like when the session started successfully. It contains the same information as the now deprecated create operation.

http://127.0.0.1:8080/api/v0/sessions/1/operations/start/1

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Clear Send

Status 200 OK Loading time: 8 ms

Request headers
 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
 Content-Type: text/plain; charset=utf-8
 Accept: */*
 Accept-Encoding: gzip, deflate, sdch
 Accept-Language: en-US,en;q=0.8
 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464

Response headers
 Date: Wed, 07 Oct 2015 14:45:20 GMT
 Content-Length: 105
 Content-Type: application/json
 Server: CherryPy/3.6.0

Raw JSON Response

Copy to clipboard Save as file

```
{
  status: "Successful"
  actionName: "start"
  state: "finished"
  sessionId: 1
}
```

The following figure shows the unsuccessful start operation because of maximum number of instances:

http://127.0.0.1:8080/api/v0/sessions/2/operations/start/2

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Clear Send

Status: 200 OK Loading time: 6 ms

Request headers:

- User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
- Content-Type: text/plain; charset=utf-8
- Accept: /*
- Accept-Encoding: gzip, deflate, sdch
- Accept-Language: en-US,en;q=0.8
- Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464

Response headers:

- Date: Wed, 07 Oct 2015 14:48:43 GMT
- Content-Length: 179
- Content-Type: application/json
- Server: CherryPy/3.6.0

Raw JSON Response

Copy to clipboard Save as file

```
{
  status: "Successful"
  actionName: "start"
  state: "finished"
  errorMessage: "Already running maximum allowed copies of IxLoad."
  sessionId: 2
}
```

Deleting a session

Deleting an IxLoad session is done in the same way that was described for generic lists. A DELETE request can be sent either to the sessions list URL, or to an actual session URL. If the request is sent to the sessions URL, all sessions will be closed. If the request is sent to a specific session's object ID, only that session will be closed.

When deleting a session, the IxLoad process underneath it will be closed.

CHAPTER 8 IxLoad Data Model

Using REST API, you can browse the IxLoad data model to retrieve or modify the current configuration. The following link contains information on where to find in the data model the following resources: L47 plugins, L23 ranges, and timelines. In addition, the link contains supported operations, such as loading and saving configurations and running a test.

Communities

You can find all the communities in the following path:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/>

All the communities in the test are shown in this list, regardless of their role: client, server, and peer. In addition, this contains both enabled and disabled communities.

You can choose to only view client communities by performing a GET operation on the same list, but by using query strings:

- [http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList?filter="role eq client"](http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList?filter='role eq client')

Community resources:

- activities: All the activities under a community can be found in the following list:
 - [http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/\\$communityObjectID/activityList](http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/$communityObjectID/activityList)
 - An activity's command list can be found under the 'agent' resource
- port list : The ports assigned to a community can be found on the 'network' resource under the community resource:
 - <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/0/network/portList>
- IP ranges : The IP ranges used by the community can be found under the 'network' resource:
 - <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/0/network/stack/childrenList/1/childrenList/1/rangeList>
 - 'stack' is the entry point in the L23 data model.

Timelines

All the timelines used in the test shows under a single list, located on the 'activeTest' resource:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/timelineList>

Plugins do not expose in REST a direct reference to their timeline (the activities do not have a 'timeline' option exposed). They have a 'timelineId' option. This option contains the 'objectID' of the required timeline in the test timeline list. If you want to change the timeline used by a certain plugin, perform a PATCH request on the activity with the following payload:

```
{"timelineId": "object ID of the desired timeline in the test timeline list"}
```

Login name

You can change the login name that can be used by their session when running, by changing the **loginName** field on the chassis chain resource:

- PATCH on <http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/>
- payload : { "loginName" : "NewLoginName" }

Modifying the activity user objective value on the fly

While the test is running, you can change the user objective value for an activity by performing a PATCH request on a URL similar to the following:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/0/activityList/0>

with the following payload:

```
{"userObjectiveValue": 100}
```

CHAPTER 9 Chassis Chain/Port Assignment Operations

Through IxLoad REST API, you can perform the following chassis and port operations:

- Add or remove a chassis
- Connect to a chassis
- Assign or unassign ports

The chassis list can be found on the 'chassisChain' root object, at the following URL:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList>

Adding a chassis

Add a chassis as follows:

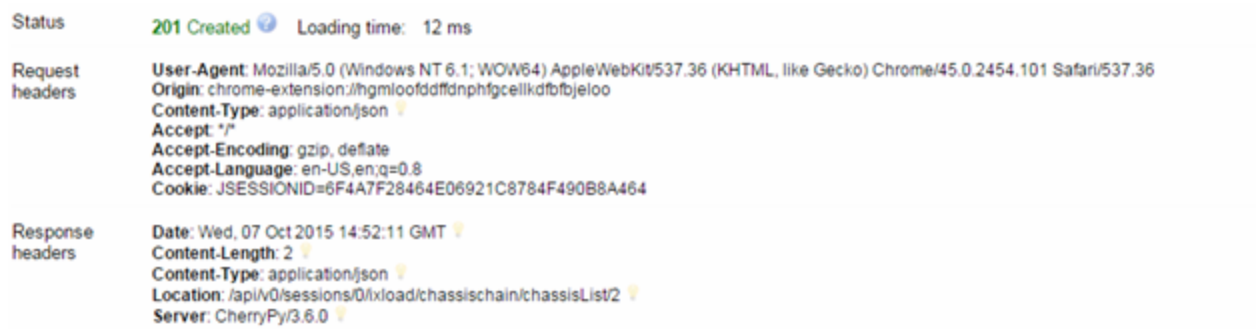
```
POST@ api/v0/sessions/0/ixload/chassischain/chassisList with {"name":"chassis ip or name" }
```

The following figure shows the input for the REST client. The newly added chassis is not connected and it has no cards or ports.

The screenshot shows a REST client interface with the following elements:

- URL:** `http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList`
- Method:** Radio buttons for GET, POST (selected), PUT, PATCH, DELETE, HEAD, OPTIONS, and Other.
- Request Body:** A text area containing the JSON payload: `{"name": "10.215.170.77"}`. Above the text area are buttons for "Raw", "Form", and "Files (0)".
- Content-Type:** A dropdown menu set to "application/json" with a note: "Set 'Content-Type' header to overwrite this value."
- Buttons:** "Clear" and "Send" buttons at the bottom right.

The response for this is shown in the following figure. The result is 201 Created.

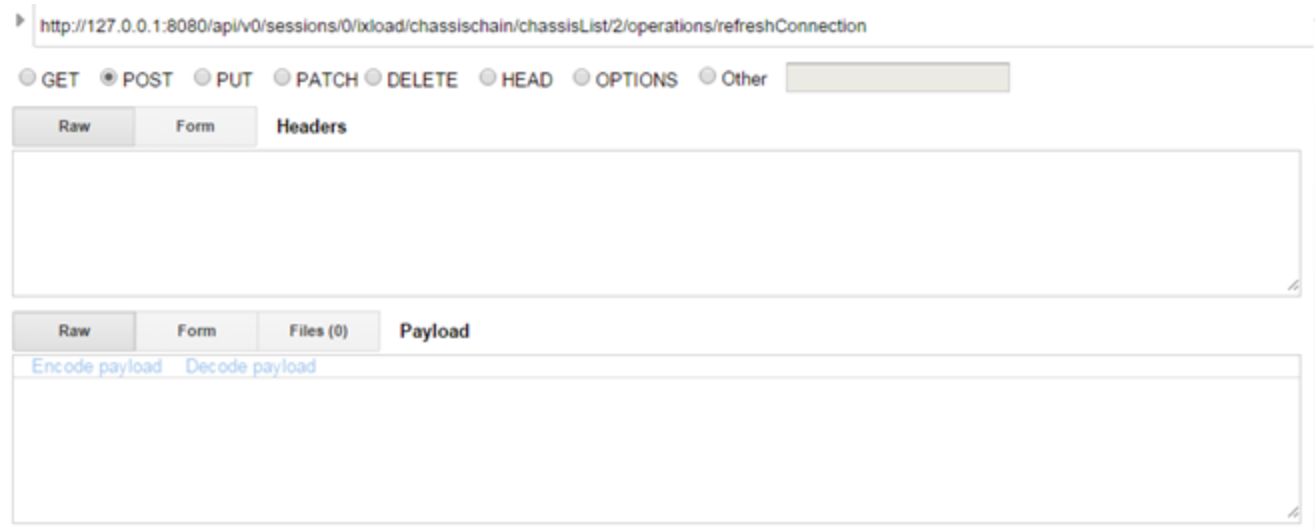


Connecting to a chassis

To connect to a chassis, do the following:

```
POST @  
api/v0/sessions/0/ixload/chassischain/chassisList/2/operations/refreshConnection
```

No payload is required. The following figure shows how this looks in the REST client:









Status should be 202 Accepted as shown in the following figure:

Status	202 Accepted  Loading time: 14 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36 Origin: chrome-extension://hgmloofddfnphfcgellkdfbfbjeloo Content-Type: application/json  Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.8 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
Response headers	Date: Wed, 07 Oct 2015 14:58:32 GMT  Content-Length: 2  Content-Type: application/json  Location: api/v0/sessions/0/ixload/chassischain/chassisList/2/operations/refreshConnection/0  Server: CherryPy/3.6.0 

The result to the refresh operation is as follows:

GET
 POST
 PUT
 PATCH
 DELETE
 HEAD
 OPTIONS
 Other

Status	200 OK  Loading time: 16 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36 Content-Type: text/plain; charset=utf-8  Accept: */* Accept-Encoding: gzip, deflate, sdch Accept-Language: en-US,en;q=0.8 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
Response headers	Date: Wed, 07 Oct 2015 15:03:18 GMT  Content-Length: 138  Content-Type: application/json  Server: CherryPy/3.6.0 

```

{
  status: "Successful"
  actionName: "refreshConnection"
  state: "finished"
  refreshedChassis: "10.215.170.77"
}
  
```

Status **200 OK** Loading time: 16 ms

Request headers
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
Content-Type: text/plain; charset=utf-8
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464

Response headers
Date: Wed, 07 Oct 2015 15:09:06 GMT
Content-Length: 297
Content-Type: application/json
Server: CherryPy/3.6.0

Raw	JSON	Response
Copy to clipboard Save as file		
<pre>{ status: "Successful" actionName: "refreshConnection" state: "finished" warning: "Could not connect to chassis 10.205.29.21. If any ports were assigned to the network they have been removed. Please reassign if chassis will be back up." refreshedChassis: "" }</pre>		

There is a new field inserted that is named **refreshedChassis**. This is referring to the ip or hostname of the chassis that was refreshed.

Usually, this contains the chassis that was refreshed. The only exception is when the loaded rxf has more than one chassis and all of them are not refreshed. In this case, **refreshedChassis** holds all the chassis in the rxf because the whole chassis chain is getting refreshed.

There are cases when the rxf has older chassis that no longer exist. To handle this, in the **refreshConnection** operation, a warning field was added that instructs the user on what happened and the **refreshedChassis** field contains only those chassis that were successfully connected to. In the preceding figure, a GET on the status of the **refreshConnection** operation shows that no chassis were refreshed and a warning message appears informing the user about what happened.

Removing a chassis

This is done by performing a simple delete operation on the chassis list. To remove all the chassis in the list, the DELETE request must be performed on the chassis list URL.

To remove only a certain chassis, the DELETE request must be performed on the following URL:

```
api/v0/sessions/0/ixload/chassischain/chassisList/chassisObjectId
```

This is consistent with DELETE operations on other IxLoad Data Model lists.

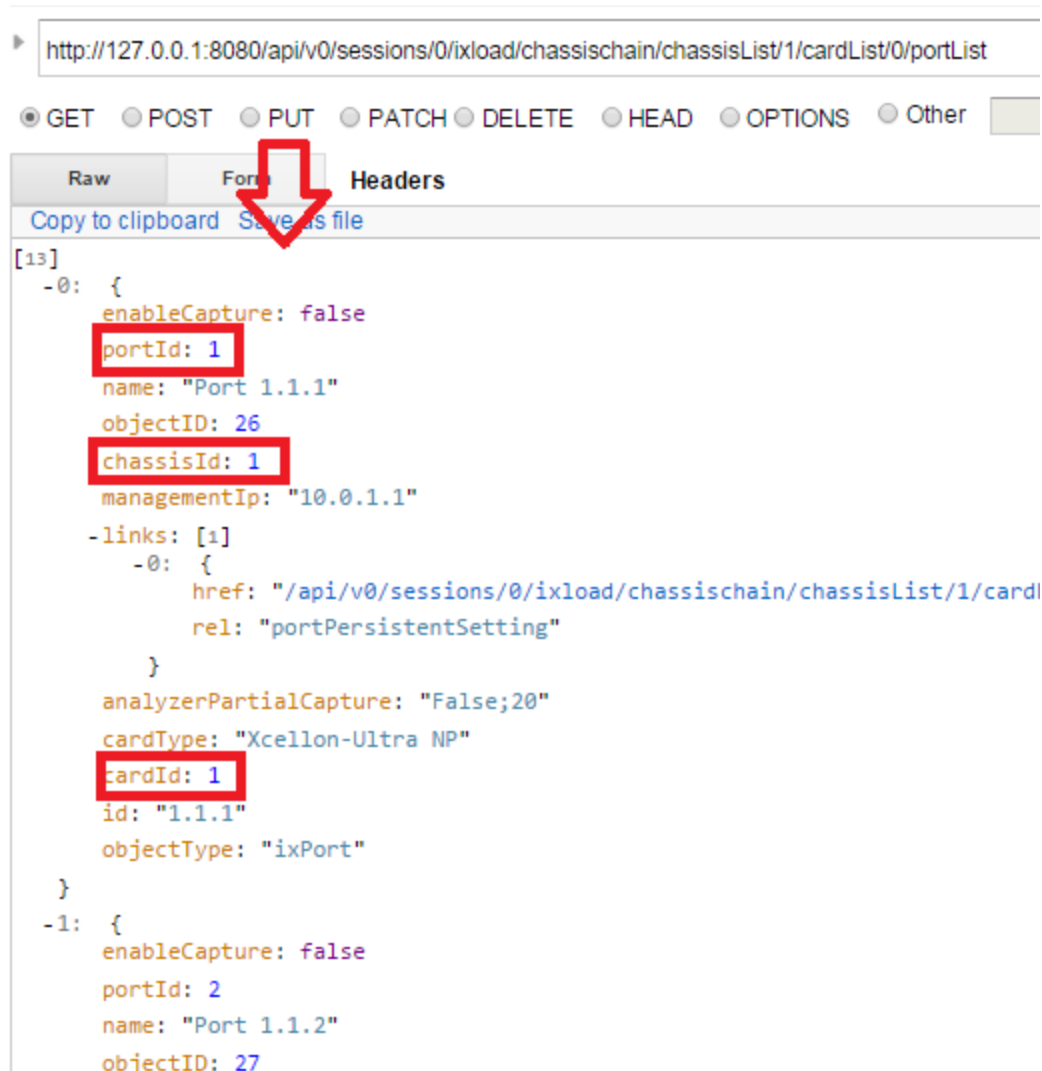
Unassign ports

Unassigning ports is done by performing a DELETE request on the network port list. This is done the same as for removing chassis - you can unassign either one of the ports (by using the port object ID), or all the ports, by performing the DELETE operation on the list URL.

Assign ports

Assigning ports is done by performing POST operations on the network port list. The POST request must receive three parameters: `chassisId`, `cardId`, and `portId`. These parameters do not represent the unique objectIDs used by REST API to identify resources as part of a list. These three parameters have the same meaning they have from UI and TCL/Python/Perl scripting, where a port is identified by using a string such as "1.1.1".

The `chassisId`, `cardId`, and `portId` can be seen on performing a GET request on the `portList` for each card of a chassis as shown in the following figure:

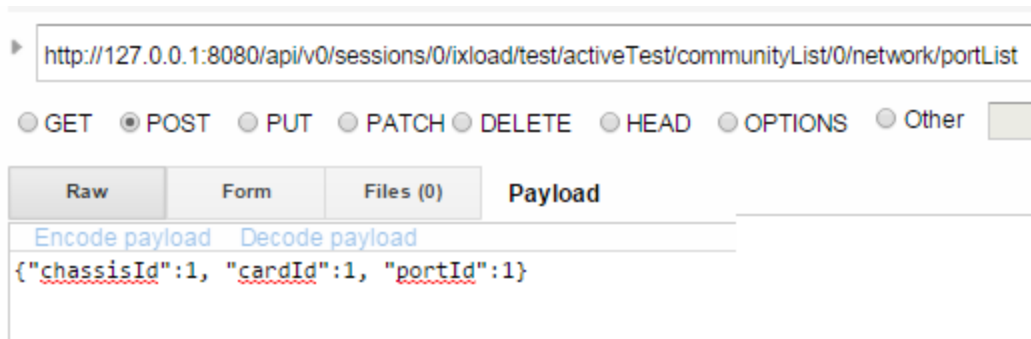


```

http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList/1/cardList/0/portList
GET POST PUT PATCH DELETE HEAD OPTIONS Other
Raw Form Headers
Copy to clipboard Save as file
[13]
-0: {
  enableCapture: false
  portId: 1
  name: "Port 1.1.1"
  objectID: 26
  chassisId: 1
  managementIp: "10.0.1.1"
  -links: [1]
    -0: {
      href: "/api/v0/sessions/0/ixload/chassischain/chassisList/1/card
      rel: "portPersistentSetting"
    }
  analyzerPartialCapture: "False;20"
  cardType: "Xcellon-Ultra NP"
  cardId: 1
  id: "1.1.1"
  objectType: "ixPort"
}
-1: {
  enableCapture: false
  portId: 2
  name: "Port 1.1.2"
  objectID: 27

```

The values highlighted in the preceding figure are the ones that are used when assigning the port as shown in the following figure:



IxVM chassis (ixChassisBuilder)

Use the chassisBuilder object to configure and manage IxVM virtual chassis, and the cards and ports on them.

To get the root chassisBuilder object, send a GET request to the following URL:

`http://serverAddress:8080/api/v0/sessions/{sessionId}/ixload/chassisBuilder`

A response will be returned in the following form, which indicates the connected chassis:

```
{
  "restObjectType": "ixChassisBuilder"
  "chassisName": "10.215.122.90"
  -"links": [1]
    -0: {
      "href": "/api/v0/sessions/0/ixload/chassisBuilder/docs"
      "rel": "docs"
    }
}
```

To display the list of operations available, send the following request:

`http://serverAddress:8080/api/v0/sessions/{sessionId}/ixload/chassisBuilder/operations`

```

{
  -"deleteCard": {
    "cardId": ""
  }
  -"updateChassisSettings": {
    "enableLicenseCheck": null
    "ntpServer": null
    "licenseServer": null
    "txDelay": null
  }
  "getChassisSettings": {}
  "hardChassisReboot": {}
  -"getCardPorts": {
    "cardId": ""
  }
  -"updatePortById": {
    "promiscMode": null
    "portId": ""
    "cardId": ""
    "lineSpeed": null
    "mtu": null
  }
  -"updateCard": {
    "cardServerId": ""
    "managementIp": null
    "keepAliveTimeout": null
  }
}

```

To execute an operation, send a POST request with the operation URL:

```

POST
http://serverAddress:8080/api/v0/sessions/
{sessionId}/ixload/chassisBuilder/operations/getChassisSettings

```

You can retrieve the operation's status by sending a GET with operation's ID:

```

GET
http://serverAddress:8080/api/v0/sessions/
{sessionId}/ixload/chassisBuilder/operations/getChassisSettings/{operationId}

```

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Status: 200: OK ? Loading time:312ms

Response headers (4) Request headers (5)

Date: Mon, 21 Mar 2016 15:34:49 GMT
 Content-Length: 272
 Content-Type: application/json
 Server: CherryPy/3.6.0

Raw JSON Response

[COPY TO CLIPBOARD](#) [SAVE AS FILE](#)

```

{
  "status": "Successful"
  "actionName": "getChassisSettings"
  "state": "finished"
  -"links": [1]
    -0: {
      "href": "/api/v0/sessions/0/ixload/chassisBuilder/operations/getChassisSettings/0/result"
      "rel": "result"
    }
}
  
```

You can retrieve the operation's result by sending the following URL:

```

GET
http://serverAddress:8080/api/v0/sessions/
{sessionId}/ixload/chassisBuilder/operations/getChassisSettings/{operationId}/
result }
  
```

The result is specified in the links dictionary from the action status URL.

The result is in the following form:

```
{
  "EnableLicenseCheck": 1
  -"links": [1]
    -0: {
      "href": "/api/v0/sessions/0/ixload/chassisBuilder/operations/getChassisSettings/0/result/docs"
      "rel": "docs"
    }
  "NtpServer": "10.215.170.157"
  "TxDelay": "1"
  "restObjectType": "ixChassisSettings"
  "LicenseServer": "10.215.122.90"
}
```

This page intentionally left blank.

CHAPTER 10 Statistics

The REST statistics component behaves similar to the StatCollectorUtils component used in TCL. You can get the available statistics for the activities configured in a test. You can also apply filters on port, net traffic, and activity.

The user is responsible to poll statistics from the web server. We will hold all statistics configured in the test in a circular buffer for a default amount of polls of 20 timestamps. As a part of the current release, the number of default polls is not configurable.

Viewing statistics

Using IxLoad REST API, you can configure L47 statistics that will be available at run time.

IxLoad REST API offers support to configure and poll statistics for L47 protocols.

The root resource for statistics in REST mode is found at the following URL:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats>

When a repository is loaded that contains L47 protocols, a GET request on this URL returns a list of statistics sources, as seen in the following figure:

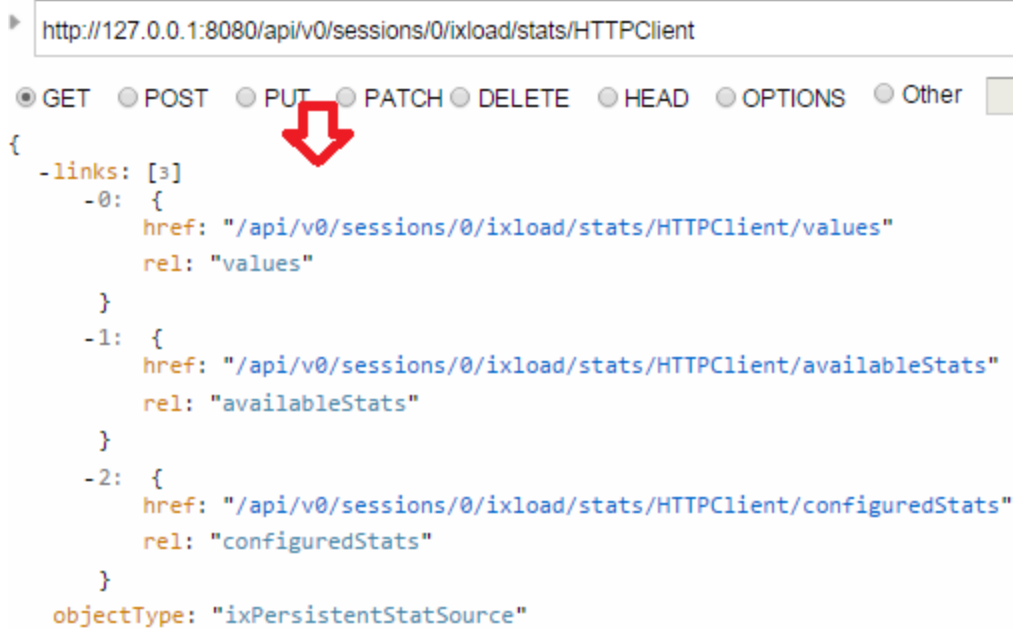


```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/

GET POST PUT PATCH DELETE HEAD OPTIONS Other

{
  -links: [4]
  -0: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPServerPerURL"
    rel: "HTTPServerPerURL"
  }
  -1: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPServer"
    rel: "HTTPServer"
  }
  -2: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClientPerURL"
    rel: "HTTPClientPerURL"
  }
  -3: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient"
    rel: "HTTPClient"
  }
  objectType: "ixRestStatController"
}
```

A GET request on any of the returned statistics sources except RunState returns three lists: availableStats, configuredStats, and values as shown in the following figure:



```

http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient

GET POST PUT PATCH DELETE HEAD OPTIONS Other

{
  -links: [3]
    -0: {
      href: "/api/v0/sessions/0/ixload/stats/HTTPClient/values"
      rel: "values"
    }
    -1: {
      href: "/api/v0/sessions/0/ixload/stats/HTTPClient/availableStats"
      rel: "availableStats"
    }
    -2: {
      href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats"
      rel: "configuredStats"
    }
  objectType: "ixPersistentStatSource"
}

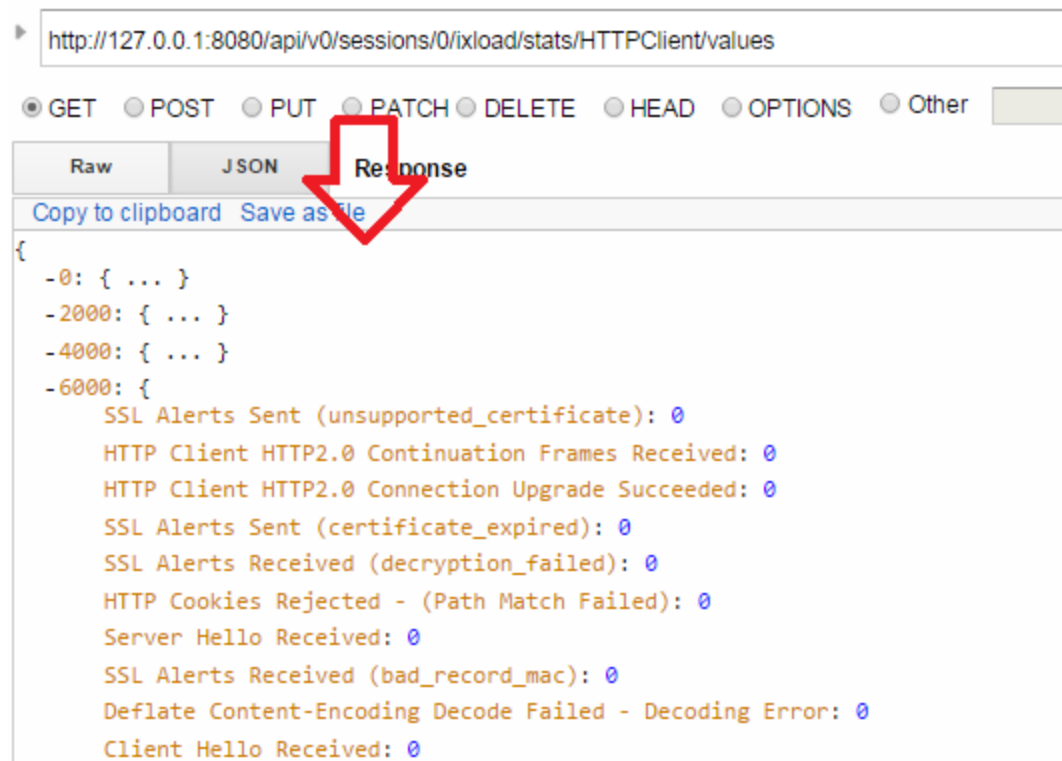
```

In the preceding figure, there are several complex objects available that refer to how statistics work in the IxLoad REST framework:

- availableStats: This is the list that contains all the available statistics for the current test. This list is read-only. You cannot remove the available statistics.
- configuredStats: This is the list that contains the statistics that have been configured for the current test. Here, you can choose to enable, disable, remove, or modify existing statistics. By default, configuredStats includes all availableStats.
- Each configured statistics resource has the following fields:
 - filterList
 - enabled
 - caption: This must be unique in the list
 - objectID: This must be unique
 - aggregationType
 - statName
- values: This is a dictionary that contains the actual statistics values during the IxLoad test run.
 - If a GET request is performed on 'values' before the test actually runs, an empty dictionary is returned.
 - The format for the dictionary is as follows: {timestamp : { stat name : stat value } }

The 'values' dictionary will only keep, at all times, the last 20 timestamps. If you do not poll the statistics in due time, you might lose some timestamps.

The following figure shows the values obtained when running a query on the HTTP client statistics values:



RunState stat source

The RunState statistics source is listed for all agents under a single statistics source called 'RunState.' There are no configurable options for the RunState statistics source. You can only perform 'GET' requests on it. The only option for RunState is the 'values' option. It does not have the 'availableStats' or 'configuredStats' options.

The URL for the RunState statistics source is as follows:

```
http://IP:8080/api/v0/sessions/sessionId/ixload/stats/RunState
```

It simply contains a link to the 'values' resource. The statistics values can be viewed at the following URL:

```
http://IP:8080/api/v0/sessions/sessionId/ixload/stats/RunState/values
```

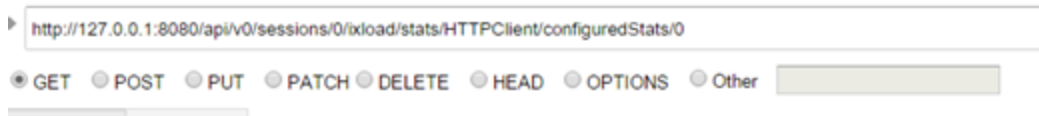
A 'GET' on the values URL before the test starts running returns an empty dictionary. After the test starts running, the dictionary is populated with the RunState statistics values for all agents.

Modifying configured statistics

Changing statistics is done by running the PATCH method on the configured statistics structure.

You can turn on or turn off statistics and change the aggregation type.

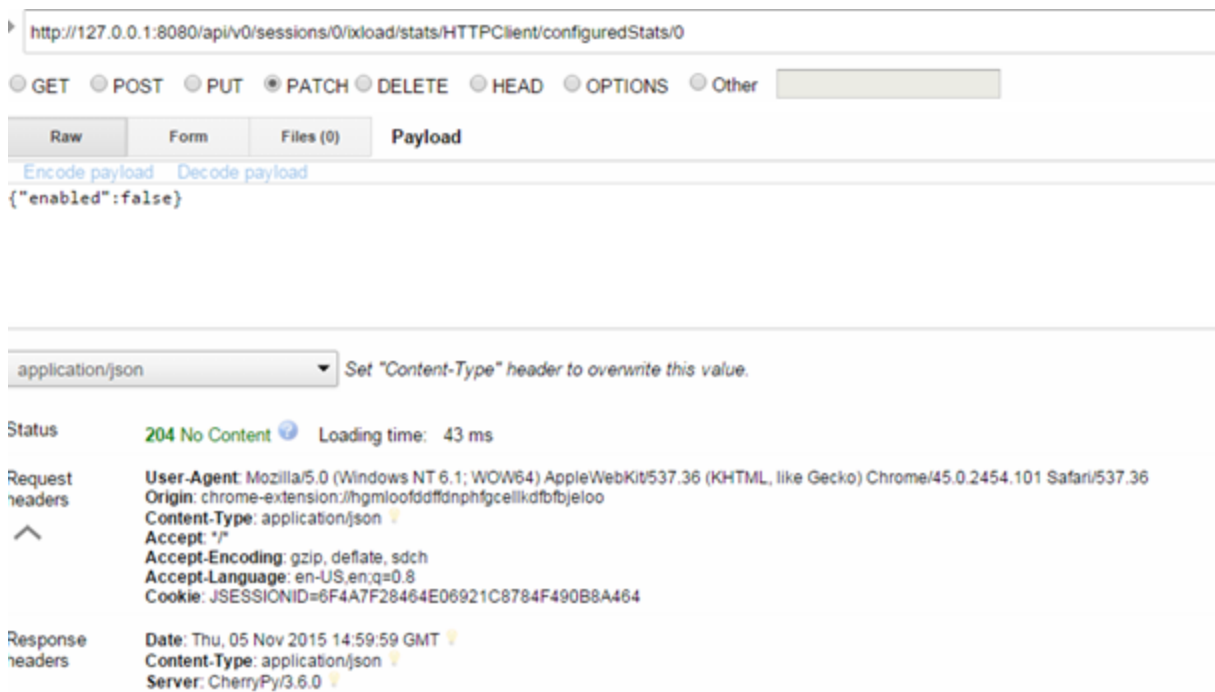
The following figure shows the URL for getting a configured statistic:



The result of get in the preceding request is shown in the following figure:



To change a configured statistic, a PATCH method is issued as shown in the following figure. The payload must contain the properties that require to be changed.



The following figure shows how the preceding PATCH method changed the configured statistics structure by turning it off:

http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0

GET
 POST
 PUT
 PATCH
 DELETE
 HEAD
 OPTIONS
 Other

Raw JSON Response

Copy to clipboard Save as file

```

{
  -links: [1]
  -0: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0/filterList"
    rel: "filterList"
  }
  enabled: false
  caption: "HTTP Simulated Users"
  aggregationType: "kSum"
  statName: "HTTP Simulated Users"
  objectType: "ixConfiguredStat"
}

```

Filtering stats

The filter statistics are accessed as shown in the following figure by issuing a GET command on the filter list from a specific configuredStat item:

http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0/filterList

GET
 POST
 PUT
 PATCH
 DELETE
 HEAD
 OPTIONS
 Other

Raw Form Headers

Raw JSON Response

Copy to clipboard Save as file

```

{
  -links: [4]
  -0: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredstats/0/filterlist/cardFilters"
    rel: "cardFilters"
  }
  -1: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredstats/0/filterlist/activityFilters"
    rel: "activityFilters"
  }
  -2: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredstats/0/filterlist/chassisFilters"
    rel: "chassisFilters"
  }
  -3: {
    href: "/api/v0/sessions/0/ixload/stats/HTTPClient/configuredstats/0/filterlist/portFilters"
    rel: "portFilters"
  }
  objectType: "ixRestFilters"
}

```

A configured statistic contains several filters, each enabling the option to get the values at:

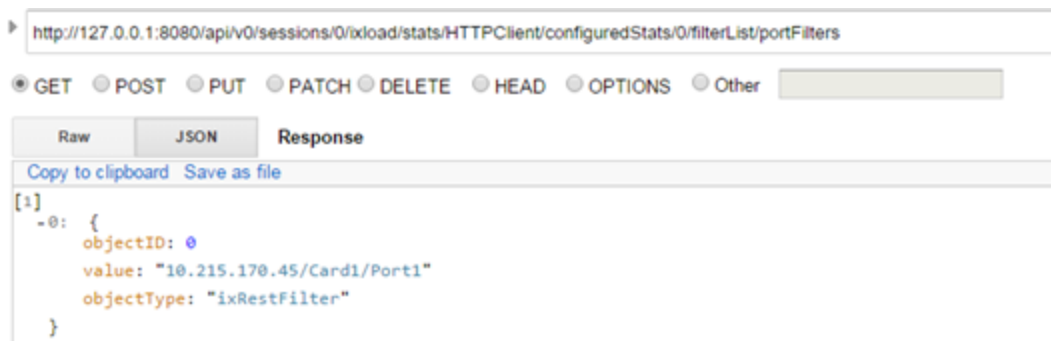
- Card level
- Activity level
- Chassis level
- Port level

Adding a port filter is demonstrated in the following figure by adding a new port to the portFilter list:

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0/filterList/portFilters`
- Method:** POST (selected)
- Headers:** (Empty)
- Payload:** `{"value": "10.215.170.45/Card1/Port1"}`
- Content-Type:** application/json
- Status:** 201 Created (Loading time: 47 ms)
- Request headers:**
 - User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
 - Origin: chrome-extension://hgml0ofddfdnphfgcellikdfbfbjeloo
 - Content-Type: application/json
 - Accept: *
 - Accept-Encoding: gzip, deflate
 - Accept-Language: en-US,en;q=0.8
 - Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
- Response headers:**
 - Date: Thu, 05 Nov 2015 15:04:12 GMT
 - Content-Length: 2
 - Content-Type: application/json
 - Location: /api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0/filterList/portFilters/0
 - Server: CherryPy/3.6.0

The following figure shows how the filter looks after it was added as per the preceding step:



Several filters can be set for multiple configured statistics according to how you will need to see the statistics. Aggregations and processing can be done in the client script after the statistics are coming in.

Adding an activity filter

Adding an activity filter to a statistic can be done by executing a POST request on a URL similar to the following:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0/filterList/activityFilters
```

with the following payload:

```
{"value": "Traffic1@Network1 - HTTPClient1"}
```

where `Traffic1@Network1` is the net traffic name (formed by the traffic and the network name) and `HTTPClient1` is the activity name.

Generated CSVs

During the IxLoad test run, CSVs files are also generated. If you do not change any settings regarding the CSV path, they are generated in the default result directory, which can be configured in IxLoad UI.

If you want to save the generated CSVs in a custom path, set the following two variables on the 'test' resource, before running the configuration:

- PATCH on `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/`
- payload:
 - `outputir : true` (the default is 'false')
 - `runResultDirFull : "F:\\path\\to\\the\\new\\result\\dir"`

CHAPTER 11 Logging

You can retrieve log messages by accessing a URL. The logs available from the REST API will be equivalent to the entries seen in the IxLoad UI. The URL where log entries are accessible is the following:

- <http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/logs>

A GET applied to the logs URL returns a list of the last log entries. By default, the last 100 entries are shown, but this number can be changed from the 'preferences' URL. Each log entry contains the moduleName, severity, timestamp, and message.

```
[
  {
    "moduleName": "ixChassisChain ",
    "severity": "Info",
    "objectID": 3,
    "timeStamp": "2016/06/02 19:03:35.838",
    "links": [
      {
        "href": "/api/v0/sessions/0/ixload/test/logs/3/docs",
        "rel": "docs"
      }
    ]
  },
  {
    "moduleName": "ixChassisChain ",
    "severity": "Info",
    "objectID": 4,
    "timeStamp": "2016/06/02 19:03:36.852",
    "links": [
      {
        "href": "/api/v0/sessions/0/ixload/test/logs/4/docs",
        "rel": "docs"
      }
    ]
  }
],
{
  "restObjectType": "ixRestLogEntry",
  "message": "Validating that 10.215.122.22 accepts incoming connections. Will try to connect for 10 seconds."
},
{
  "restObjectType": "ixRestLogEntry",
  "message": "Validation for 10.215.122.22 is completed. IP is valid and connection succeeded."
},
]
```

This page intentionally left blank.

CHAPTER 12 REST Script Templates

Scripts that create and use an IxLoad session in REST mode can be written in any language that supports performing REST requests. An installed IxLoad build contains a set of sample scripts that perform basic IxLoad operations from REST by using Python. The samples are found in the root installation folder of IxLoad in a subfolder named RestScripts.

The sample folder contains four Python scripts. The four scripts located in the samples folder need to be run with a Python executable that has the `requests` and `pyOpenSSL` modules installed, as described in the README.txt file included in the RestScripts folder. The scripts do not require any command line arguments. They can be simply executed by performing `python.exe SimpleRun.py`.

Before running the scripts, change the configuration data (IxLoad Version, chassis, rxf file path) in the beginning of each script accordingly.

The provided templates act as guidance. The IxLoad REST API is compatible with any programming language that supports running HTTP requests.

The REST script templates rely on two utility Python files:

- `IxLoadUtils.py`, which deals with specific IxLoad REST API functionality.
- `IxLoadRestUtils.py`, which deals with providing the underlying abstraction level that `IxLoadUtils` uses to receive, interpret, and dispatch requests.

The two files are helpers to implement a Python script to handle REST communication with the IxLoad REST framework. The template files expose basic workflow scenarios as examples for users to understand how to automatically configure IxLoad through REST.

AddNewCommand

This template does the following:

- Create a session.
- Load an Rxf.
- Clear the chassis list.
- Add a chassis.
- Assign ports to the networks.
- Clear the command list for client activity.
- Update the command list of the client HTTP activity by doing the following:
 - Adding a GET command with custom properties.
 - Adding a POST command with custom properties.
- Save the Rxf.

- Start the test.
- Poll the stats.
- Close the IxLoad session.

ChangeAgentObjectives

This template does the following:

- Create a session.
- Load an Rxf.
- Clear the chassis list.
- Add a chassis.
- Assign ports to the networks.
- Update the activity options by doing the following:
 - Enabling constraints.
 - Setting a constraint value.
 - Changing the objective type.
 - Setting a new objective type.
- Save the Rxf.
- Start the test.
- Poll the stats.
- Close the IxLoad session.

ChangeIpType

This template does the following:

- Create a session.
- Load an Rxf.
- Clear the chassis list.
- Add a chassis.
- Assign ports to the networks.
- Update the ip Ranges changing the count and the ipAddress.
- Save the Rxf.
- Start the test.
- Poll the stats.
- Close the IxLoad session.

SimpleRun

This template does the following:

- Create a session.
- Load an Rxf.
- Clear the chassis list.

- Add a chassis.
- Assign ports to the networks.
- Save the Rxf.
- Start the test.
- Poll the stats.
- Close the IxLoad session.

IxLoadRestUtils

This module defines the following:

class Connection(__builtin__.object)

This is the class that executes the HTTP requests to the application instance. It handles creating the HTTP session and executing HTTP methods.

Methods defined here are as follows:

- `__init__(self, siteUrl, apiVersion)`

Args:

- `siteUrl` is the actual URL to which the Connection instance will be made.
- `apiVersion` is the actual version of the REST API that the Connection instance will use.
- The HTTP session will be created when the first http request is made.

- `httpDelete(self, url="", data="", params={}, headers={})`

Method for calling HTTP DELETE. Will return the HTTP reply.

- `httpGet(self, url="", data="", params={}, headers={})`

Method for calling HTTP GET. This will return a WebObject that has the fields returned in JSON format by the GET operation.

- `httpPatch(self, url="", data="", params={}, headers={})`

Method for calling HTTP PATCH. Will return the HTTP reply.

- `httpPost(self, url="", data="", params={}, headers={})`

Method for calling HTTP POST. Will return the HTTP reply.

- `httpRequest(self, method, url="", data="", params={}, headers={})`

Args:

- `Method` (mandatory) represents the HTTP method that will be executed.
- `url` (optional) is the URL that will be appended to the application URL.
- `data` (optional) is the data that needs to be sent along with the HTTP method as the JSON payload.
- `params` (optional) is the payload python dictionary (not necessary if data is used).

- headers (optional) are the HTTP headers that will be sent along with the request. If left blank, will use default.

Method for making a HTTP request. The method type (GET, POST, PATCH, DELETE) will be sent as a parameter. Along with the url and request data. The HTTP response is returned.

Class methods defined here are as follows:

- `urljoin(cls, base, end)` from `__builtin__.type`

Join two URLs. If the second URL is absolute, the base is ignored. Use this instead of `urlparse.urljoin` directly so that we can customize its behavior if necessary.

Currently differs in that it

1. Appends a/to base if not present.
2. Casts end to a str as a convenience.

Data descriptors defined here are as follows:

- `__dict__`
dictionary for instance variables (if defined)
- `__weakref__`
list of weak references to the object (if defined)

Data and other attributes defined here are as follows:

- `kContentJson = 'application/json'`
- `kHeaderContentType = 'content-type'`

class WebList(__builtin__.list)

By using this class, a JSON list is transformed in a list of WebObject instances.

Methods defined here are as follows:

- `__init__(self, entries=[])`
Create a WebList from a list of items that are processed by the `_WebObject` function.

Data descriptors defined here are as follows:

- `__dict__`
dictionary for instance variables (if defined)
- `__weakref__`
list of weak references to the object (if defined)

class WebObject(__builtin__.object)

A WebObject instance will have its fields set to correspond to the JSON format received on a GET request. For example, a response in the format: {"caption": "http"} returns an object that has obj.caption="http."

Methods defined here are as follows:

- `__init__(self, **entries)`
Create a WebObject instance by providing a dictionary having a property - value structure.
- `getOptions(self)`
Get the JSON dictionary, which represents the WebObject instance.

Data descriptors defined here are as follows:

- `__dict__`
dictionary for instance variables (if defined)
- `__weakref__`
list of weak references to the object (if defined)

Functions

- `formatDictToJSONPayload(dictionary)`
Converts a given Python dictionary instance to a string JSON payload that can be sent to a REST API.
- `getConnection(server, port)`
Gets a Connection instance, which will be used to make the HTTP requests to the application.

IxLoadUtils

The IxLoadUtils module is a collection of specific functions that deal with common IxLoad workflows.

Functions

- `addChassisList(connection, sessionUrl, chassisList)`
This method is used to add one or more chassis to the chassis list.
Args:
 - `connection` is the connection object that manages the HTTP data transfers between the client and the REST API.
 - `sessionUrl` is the address of the session that should run the test.
 - `chassisList` is the list of chassis that will be added to the chassis chain.
- `addCommands(connection, sessionUrl, commandDict)`
This method is used to add commands to a certain list of provided agents.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- commandDict is the Python dictionary that holds the mapping between agent name and specific commands. (commandDict format -> { agent name : [{ field : value }] })

- assignPorts(connection, sessionUrl, portListPerCommunity)

This method is used to assign ports from a connected chassis to the required NetTraffics.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- portListPerCommunity is the dictionary mapping NetTraffics to ports (format -> { community name : [port list] })

- changeActivityOptions(connection, sessionUrl, activityOptionsToChange)

This method will change certain properties for the provided activities.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- activityOptionsToChange is the Python dictionary that holds the mapping between agent name and specific properties (activityOptionsToChange format: { activityName : { option : value } })

- changeCardsInterfaceMode (connection, chassisChainUrl, chassisIp, cardIdList, mode)

This method is used to change the interface mode on a list of cards from a chassis. To call this method, the required chassis must be already added and connected.

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- chassisChainUrl is the address of the chassisChain resource.
- chassisIp is the IP or host name of the chassis that contains the card(s).
- cardIdList is a list of card IDs.
- mode is the interface mode that will be set on the cards. Possible options are (depending on card type): 1G, 10G, 40G, 100G

- changeIpRangesParams(connection, sessionUrl, ipOptionsToChangeDict)

This method is used to change certain properties on an IP Range.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- ipOptionsToChangeDict is the Python dict holding the items in the IP range that will be changed.
- (ipOptionsToChangeDict format -> { IP Range name : { optionName : optionValue } })

- clearAgentsCommandList(connection, sessionUrl, agentNameList)

This method clears all commands from the command list of the agent names provided.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- agentNameList the list of agent names for which the command list will be cleared.

- clearChassisList(connection, sessionUrl)

This method is used to clear the chassis list. After execution, no chassis should be available in the chassisList.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.

- createSession(connection, ixLoadVersion)

This method is used to create a new session. It will return the URL of the newly created session.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- ixLoadVersion is the actual IxLoad version to start.

- deleteSession(connection, sessionUrl)

This method is used to delete an existing session.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session to delete.

- getCommandListUrlForAgentName(connection, sessionUrl, agentName)

This method is used to get the commandList url for a provided agent name.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.
- agentName is the agent name for which the commandList address is provided.

- `getIPRangeListUrlForNetworkObj(connection, networkUrl)`

This method will return the IP Ranges associated with an IxLoad Network component.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- networkUrl is the REST address of the network object for which the network ranges will be provided.

- `getTestCurrentState(connection, sessionUrl)`

This method gets the test current state (for example, running, unconfigured).

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.

- `getTestRunError(connection, sessionUrl)`

This method gets the error that appeared during the last test run.

If no error appears (the test ran successfully), the return value is 'None.'

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.

- `loadRepository(connection, sessionUrl, rxfFilePath)`

This method performs a POST request to load a repository.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session to load the rxf for.
- rxfFilePath is the local rxf path on the computer that holds the IxLoad instance.

- `performGenericDelete(connection, listUrl, payloadDict)`

This performs a generic DELETE method on a given URL.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
 - url is the address of where the operation will be performed.
 - payloadDict is the Python dictionary with the parameters for the operation.
- performGenericOperation(connection, url, payloadDict)

This performs a generic operation on the given URL. It will wait for it to finish.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
 - url is the address of where the operation will be performed.
 - payloadDict is the python dict with the parameters for the operation.
- performGenericPatch(connection, url, payloadDict)

This performs a generic PATCH method on a given URL.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
 - url is the address of where the operation will be performed.
 - payloadDict is the Python dictionary with the parameters for the operation.
- performGenericPost(connection, listUrl, payloadDict)

This performs a generic POST method on a given URL.

Args:

- connection is the connection object.
 - url is the address of where the operation will be performed.
 - payloadDict is the python dict with the parameters for the operation.
- pollStats(connection, sessionUrl, watchedStatsDict, pollingInterval=4)

This method is used to poll the stats. Polling stats is per request but this method does a continuous poll.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
 - sessionUrl is the address of the session that should run the test.
 - watchedStatsDict these are the stats that are being monitored.
 - pollingInterval the polling interval is 4 by default but can be overridden.
- runTest(connection, sessionUrl)

This method is used to start the currently loaded test. After starting the 'Start Test' action, wait for the action to complete.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.

- saveRxf(connection, sessionUrl, rxfFilePath)

This method saves the current rxf to the disk of the computer on which the IxLoad instance is running.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session to save the rxf for.
- rxfFilePath is the location where to save the rxf on the machine that holds the IxLoad instance.

- setCardsAggregationMode(connection, chassisChainUrl, chassisIp, cardIdList, mode)

This method is used to change the aggregation mode on a list of cards from a chassis. To call this method, the required chassis must be already added and connected.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- chassisChainUrl is the address of the chassisChain resource.
- chassisIp is the IP or hostname of the chassis that contains the card(s).
- cardIdList is a list of card IDs.
- mode is the aggregation mode that will be set on the cards. Possible options are (depending on card type): NA (Non Aggregated), 1G, 10G, 40G.

- waitForActionToFinish(connection, replyObj, actionUrl)

This method waits for an action to finish executing. After a POST request is sent to start an action, the HTTP reply will contain, in the header, a 'location' field, that contains a URL.

The action URL contains the status of the action. This performs a GET on that URL every 0.5 seconds until the action finishes with a success.

If the action fails, this will show an error and print the action's error message.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.

- replyObj the reply object holding the location.
- actionUrl: the url pointing to the operation.
- waitForAllCaptureData(connection, sessionUrl)

This method is used to wait for the test to capture all the port data that was received after the test has finished running.

Args:

- connection is the connection object that manages the HTTP data transfers between the client and the REST API.
- sessionUrl is the address of the session that should run the test.

The IxLoad REST templates provide a basic way of doing common tasks with IxLoad. The same can be achieved in other programming languages, not necessarily Python.

This page intentionally left blank.

