IxLoad

REST API Programming Guide

Version 8.50EA-Update3

Notices

Copyright Notice

© Keysight Technologies 2015–2019

No part of this document may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Keysight Technologies, Inc. as governed by United States and international copyright laws.

Warranty

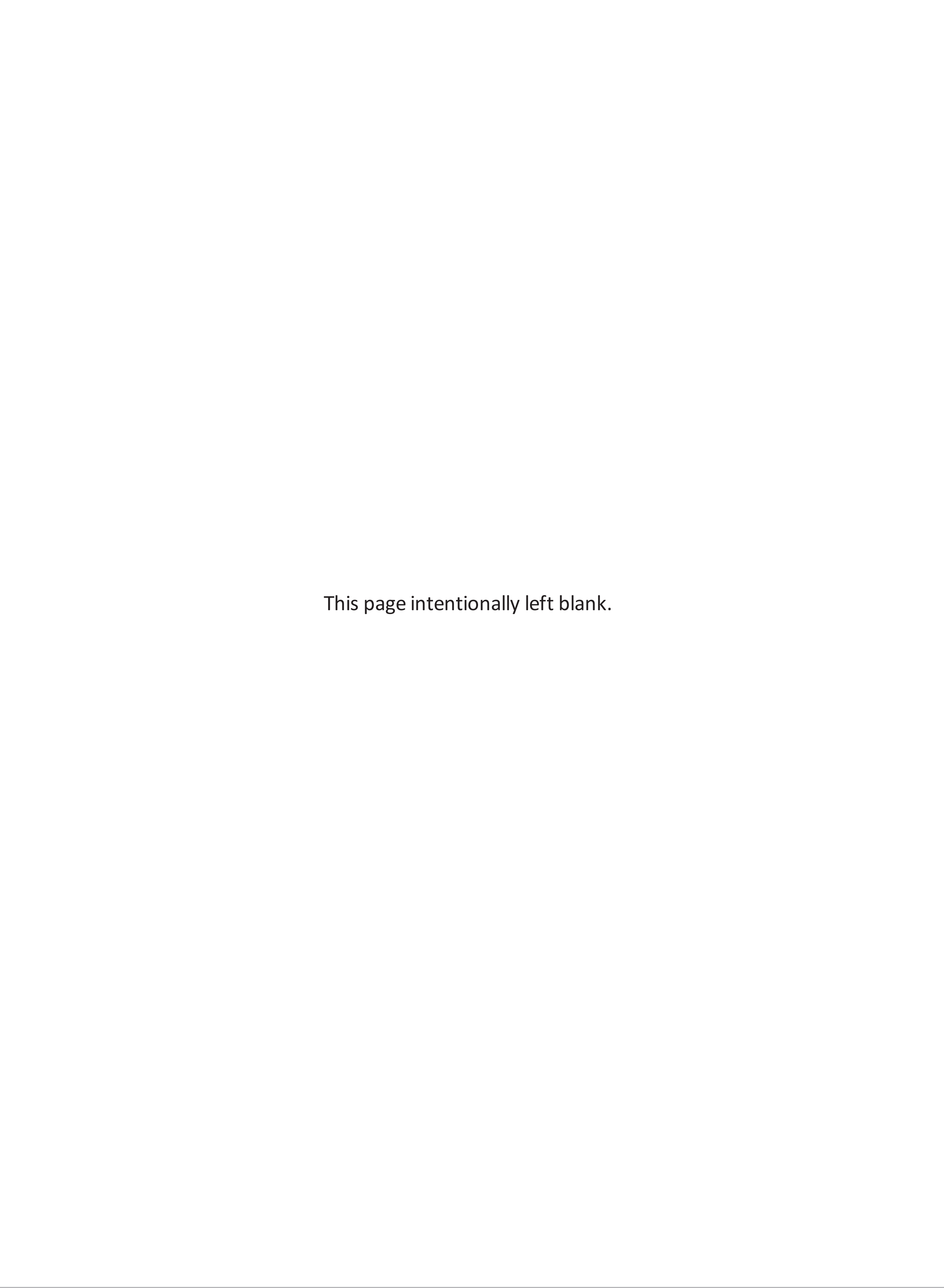
The material contained in this document is provided “as is,” and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Keysight disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Keysight shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Keysight and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at http://www.keysight.com/find/sweula or https://support.ixiacom.com/support-services/warranty-license-agreements. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data. 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.



This page intentionally left blank.

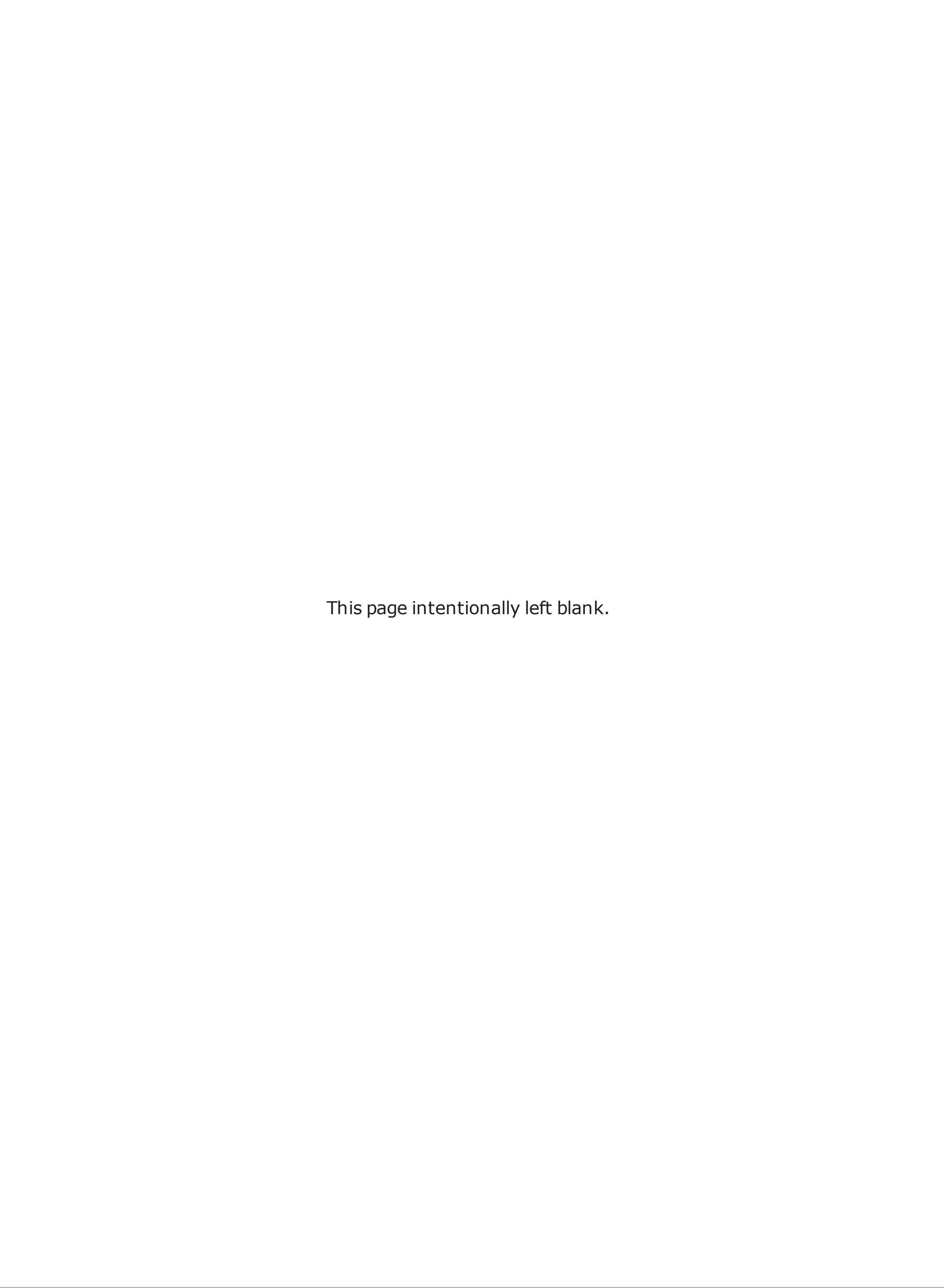
Contacting Us

Ixia headquarters

26601 West Agoura Road
Calabasas, California 91302
+1 877 367 4942 – Toll-free North America
+1 818 871 1800 – Outside North America
+1.818.871.1805 – Fax
www.ixiacom.com/contact/info

Support

Global Support	+1 818 595 2599	support@ixiacom.com
Regional and local support contacts:		
APAC Support	+91 80 4939 6410	support@ixiacom.com
Australia	+61-742434942	support@ixiacom.com
EMEA Support	+40 21 301 5699	support-emea@ixiacom.com
Greater China Region	+400 898 0598	support-china@ixiacom.com
Hong Kong	+852-30084465	support@ixiacom.com
India Office	+91 80 4939 6410	support-india@ixiacom.com
Japan Head Office	+81 3 5326 1980	support-japan@ixiacom.com
Korea Office	+82 2 3461 0095	support-korea@ixiacom.com
Singapore Office	+65-6215-7700	support@ixiacom.com
Taiwan (local toll-free number)	00801856991	support@ixiacom.com



This page intentionally left blank.

CONTENTS

Contacting Us iv

New in this Release x

Before you Begin xii

REST Resources 1

Supported Features 3

API version v1 4

Using the REST API over HTTPS 7

Self-signed certificates 7

Script changes required for HTTPS 8

Errors from REST UI clients 8

REST Authentication 9

Enabling authentication on Windows 9

Enabling authentication on Linux 10

Authenticating REST requests 11

Retrieving the api-key 12

Script changes required for authentication 12

Supporting Methods and Running Operations 15

REST representation 15

Preferences 16

IxLoad REST methods 17

GET 17

PATCH 18

POST 19
DELETE 20
OPTIONS 21
Operations **23**
Starting an operation 23
Getting an operation's status 24
Examples of common operations in the IxLoad REST API 26
Query strings 29
Collecting diagnostics 30
Deleting the results directory after running a test 32
extractDataModel operation 34
IxLoadGateway: IxLoad Session Handling **37**
Creating a new session 37
New session with a specified version 37
New session with the latest version 41
Deleting a session 42
Uploading and downloading files **43**
API Browser **45**
How to find URLs in a REST API session **48**
IxLoad Data Model **51**
Communities 51
Timelines 51
Login name 52
DUTs 52
Expiration timer 54
Enabling Analyzer and downloading captures 55
Modifying the activity user objective value on the fly 56

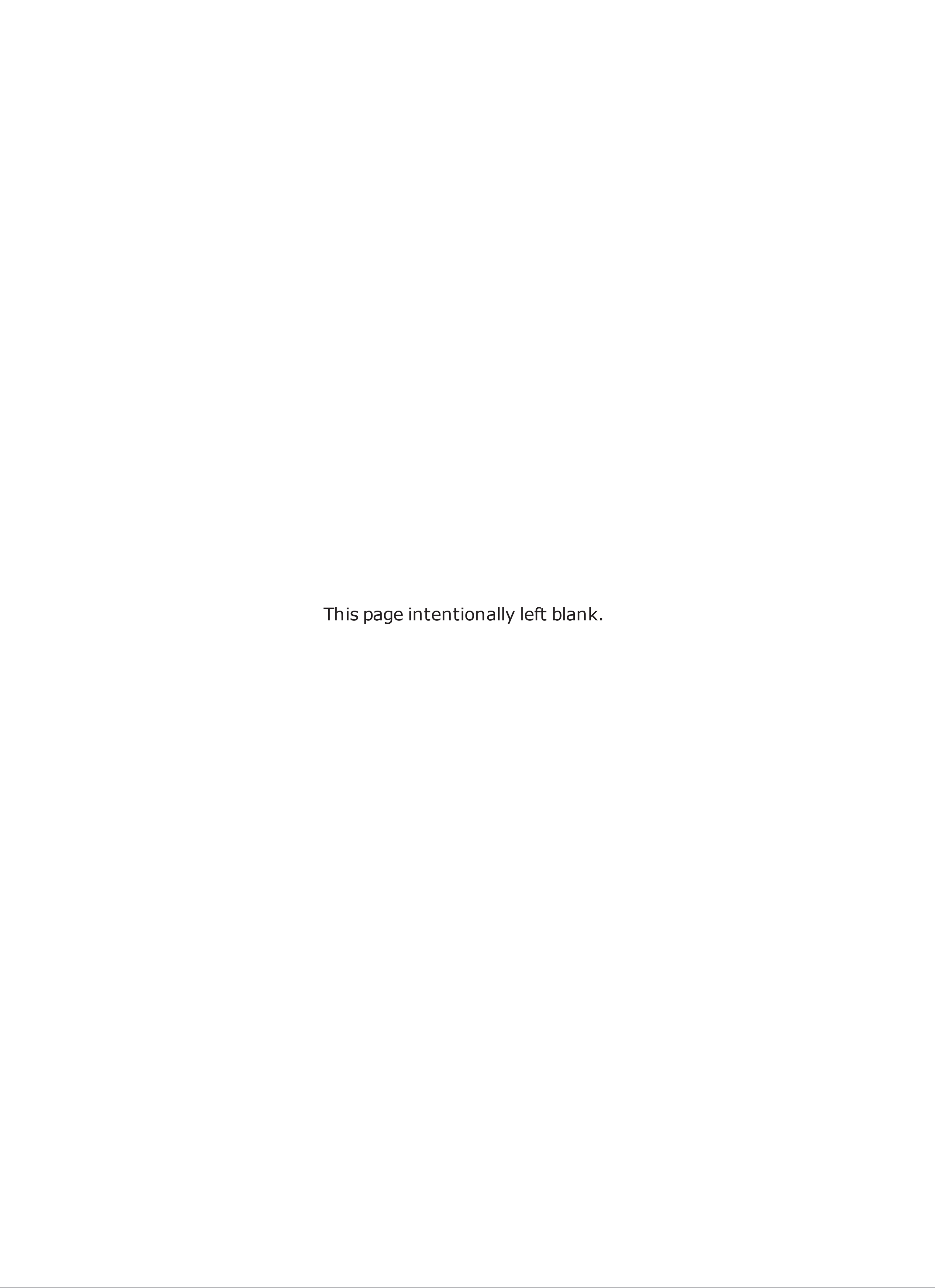
Chassis Chain/Port Assignment Operations 57
Adding a chassis 57
Connecting to a chassis 58
Removing a chassis 60
Assigning ports 60
Unassigning ports 62
IxVM chassis (ixChassisBuilder) 62
Upload and Download Diameter XML Configuration Files 65
Statistics 67
Viewing statistics 67
Statistics views 70
RunState stat source 72
Video client per-stream statistics 72
Modifying configured statistics 74
Filtering stats 76
Generated CSVs 78
Logging 79
REST Script Templates 83
AddNewCommand.py 83
ChangeAgentObjectives.py 84
ChangeIpType.py 84
CIFSfromScratch.py 85
Dhcpv4v6_config_from_scratch.py 85
DNS_with_DUT_from_scratch.py / DNS_config_from_scratch.py 85
FTP_config_from_scratch.py 85
HTTP_ssl_ipsec_ipv4v6_config_from_scratch.py 85
IMAP_config_from_scratch.py 85

POP3ConfigFromScratch.py 85
RepRunner.py 85
RTSP_config_from_scratch.py 85
SimpleRun.py 85
SimpleRunCapturesEnabled.py 86
SMTPfromScratch 86
StatelessPeerFS.py 86
TFTP_config_from_scratch.py 86
VoIPSIP_config_from_scratch.py 86
IxLoadRestUtils 87
class Connection(__builtin__.object) 87
class WebList(__builtin__.list) 88
class WebObject(__builtin__.object) 88
Functions 89
IxLoadUtils 90

New in this Release

The following features are new in this release:

Deleting the test results directory	You can delete the test results directory. See Deleting the results directory after running a test on page 32.



This page intentionally left blank.

Before you Begin

Before you begin using the REST API, review the sections below.

Authentication

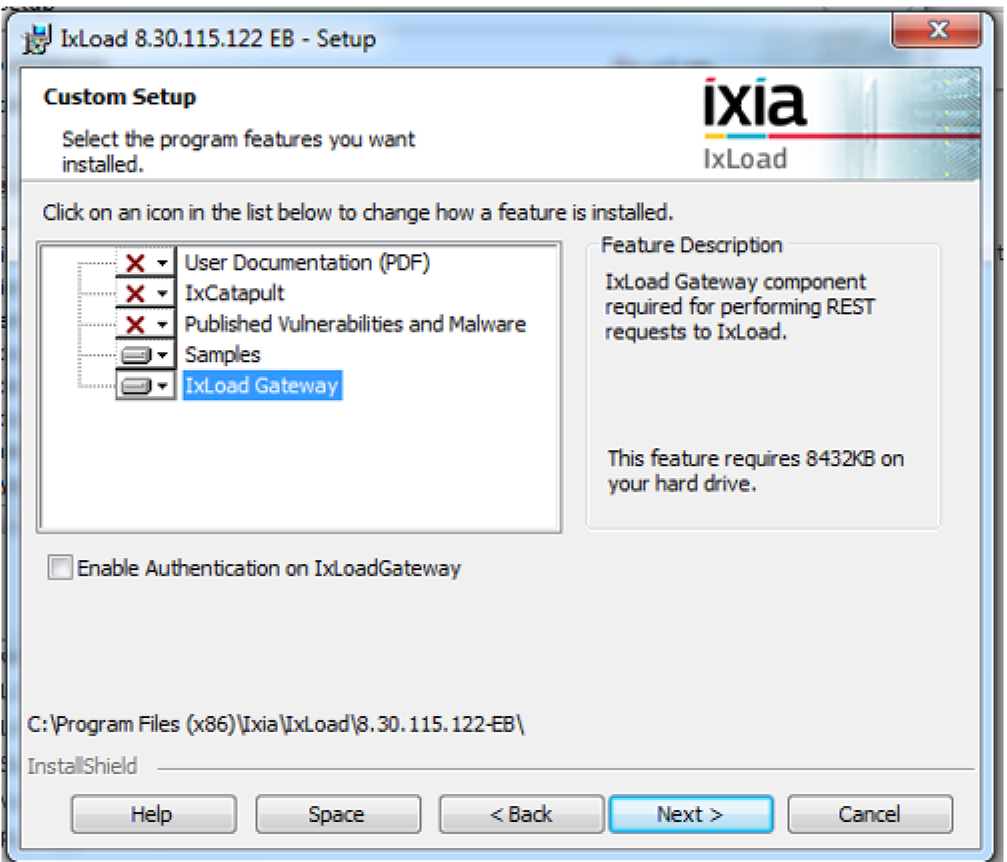
If you want to use the REST API with authentication, an Ixia User Management server must be available on the network. See REST Authentication on page 9 for more information.

Gateway Service

The IxLoad Gateway Service must 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 the Gateway Service, select **Custom Setup** during IxLoad installation.

If you have already installed IxLoad, select **IxLoad** from the list of installed applications (**Control Panel** > **Programs and Features**), right-click, and then select **Modify**. The installer runs and you can install the IxLoad Gateway Service.


IxLoad 8.30.115.122 EB - Setup
Custom Setup
Select the program features you want installed.
ixia
IxLoad
Click on an icon in the list below to change how a feature is installed.
User Documentation (PDF)
IxCatapult
Published Vulnerabilities and Malware
Samples
IxLoad Gateway
Feature Description
IxLoad Gateway component required for performing REST requests to IxLoad.
This feature requires 8432KB on your hard drive.
Enable Authentication on IxLoadGateway
C:\Program Files (x86)\Ixia\IxLoad\8.30.115.122-EB\
InstallShield
Help
Space
< Back
Next >
Cancel

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.

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 L2-3 range.
- Support L4-7 plugin.
- Save configuration modified through REST API.
- View, add, or delete the configured L2-3 and L4-7 statistics.
- Run test.
- Poll L2-3 and L4-7 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.
- Creating or editing voice scenarios
- Creating or editing Diameter scenarios (except for importing/exporting XML files, which is supported)

API version v1

Prior to IxLoad 8.50, all REST API requests were made on URLS that used API version `v0`, so the URLs started with `/api/v0`.

Beginning with 8.50, the REST API version is `v1`, so the URLs now begin with `/api/v1`.

Version `v0` is still available, and contains the same functionality as `v1`. Scripts that use `v0` will continue to function in the same way, the only things that differ are some response formats.

Version `v1` contains the following changes:

Changed "links" responses for all fields:

- Changed "`rel`" field to "`child`" for most options
- Added "`method`":"`GET`" entry for all child nodes
- Added `self` node that has URL to self
- Add `meta` node for self with “`method`”:”`OPTIONS`”

v0

```
"links": [
    {
        "href": "/api/v0/sessions/1/ixload",
        "rel": "ixload"
    },
    {
        "href": "/api/v0/sessions/1/docs",
        "rel": "docs"
    }
],
```

v1

```
"links": [
    {
        "href": "/api/v1/sessions/1/ixload",
        "method": "GET",
        "rel": "child"
    },
    {
        "href": "/api/v1/sessions/1/docs",
        "method": "GET",
        "rel": "child"
    },
    {
        "href": "/api/v1/sessions/1",
        "method": "GET",
        "rel": "self"
    },
    {
        "href": "/api/v1/sessions/1",
        "method": "OPTIONS",
        "rel": "meta"
    }
],
```

applicationTypes response

A `GET` on `api/vX/applicationtypes` will return:

v0	v1
{ "appName": "8.40.0.277" }	{ "name": "8.40.0.277" }

backendType and applicationType fields

Added backendType and applicationType fields on the session object (visible on GET on /api/vX/sessions/X).

Start session parameter

Changed the name of the parameter required to create an IxLoad session for v1. Creating a session will require the following payload:

v0	v1
{“ixLoadVersion”:”8.50.0.75”}	{“applicationVersion”:”8.50.0.75”}

204 No Content response

Starting an operation returns status 204 No Content on v1 (on v0, it returned 202 Acccepted).

Multi-POST support

Added the ability to add multiple objects to a list. Instead of a dictionary that contains the options that will define the object to be created, you can supply a list of dictionaries. This will create a new item for each dictionary in the list. This option was introduced for v1, but is available in v0 also.

Example:

Adding one chassis : {“name”:”tomini”}

Adding two chassis: [{“name”:”tomini”},{“name”:”ixro-chassis”}]

Operation status URLs

Operation status URLs return the information in a different format. When retrieving the status of an operation, the fields retrieved for v1 will be different than for v0. A get on the following URL will return the following body
api/vX/sessions/0/ixload/chassischain/chassisList/0/operations/refreshConnection/1

v0	v1

{
"status":
"Successful",
"actionName":
"refreshConnecti
on",
"state":
"finished",
"result": "",
 	
"refreshedChassi
s": "tomini"
}
	{
	"url":
	"/api/v1/sessions/0/ixload/chassischain/chassisList/0/operations/refresh
	Connection/2",
	"state": "SUCCES",
	"result": "",
	"progress": 100,
	"type": "refreshConnection",
	"id": 2,
	"refreshedChassis": "tomini"
	}
	Possible values for `state` in `v1`: `IN_PROGRESS`, `ERROR`, `SUCCESS`

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.

REST Authentication

Authentication is optional in the REST API.

To use authentication:

- an Ixia User Management server must be configured and present on the network and
- you must have an account on the User Management server.

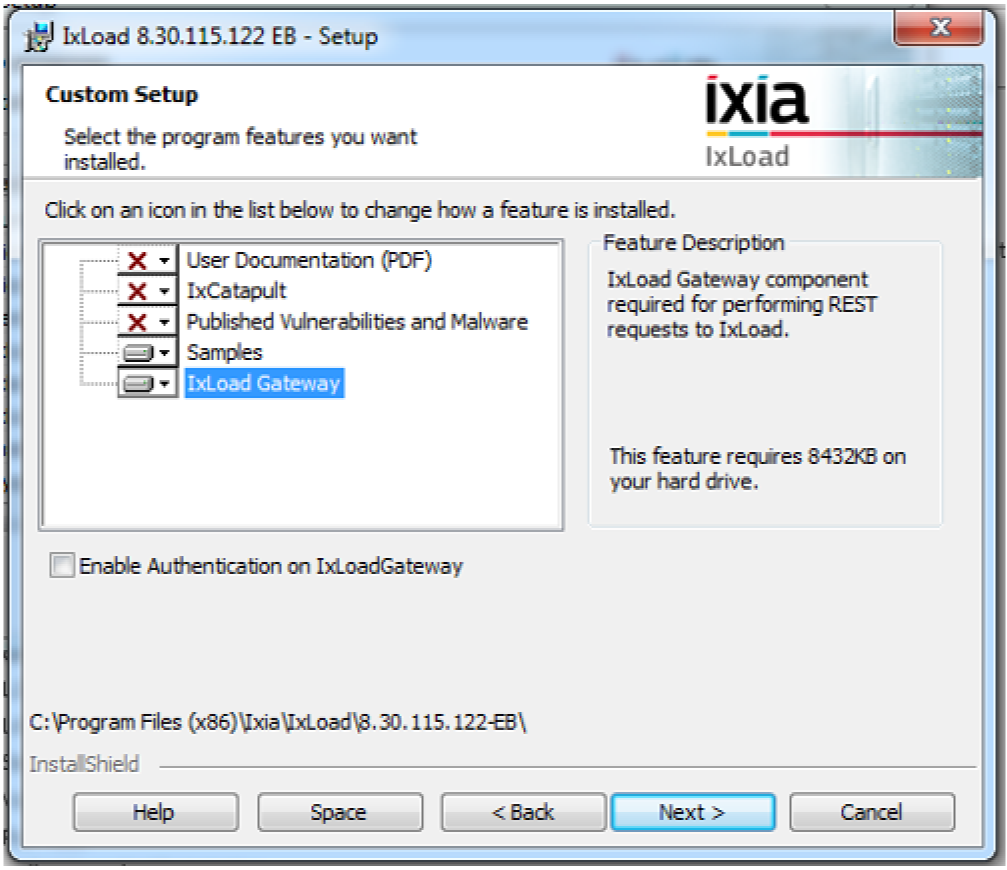
User Management is a standalone application that you can download from the IxLoad section of Ixia's website (https://support.ixiacom.com/support-overview/product-support/downloads-updates/versions/33).

After turning on authentication, most REST requests must include an `api-key` that is unique to each user. See Retrieving the api-key on page 12.

Enabling authentication on Windows

To enable REST authentication on Windows, during the IxLoad installation , select **Custom Setup** and choose the **IxLoad Gateway** feature.

To turn authentication on, select the **Enable Authentication on IxLoadGateway** check box.



Authentication can be turned on or off every time IxLoad and the IxLoad Gateway are installed.

For example, if you install one IxLoad/IxLoad Gateway version and turn on authentication, then you install a newer version and you do not select the **Enable Authentication on IxLoadGateway** check box, after the install is completed, authentication will be turned off.

Enabling authentication on Linux

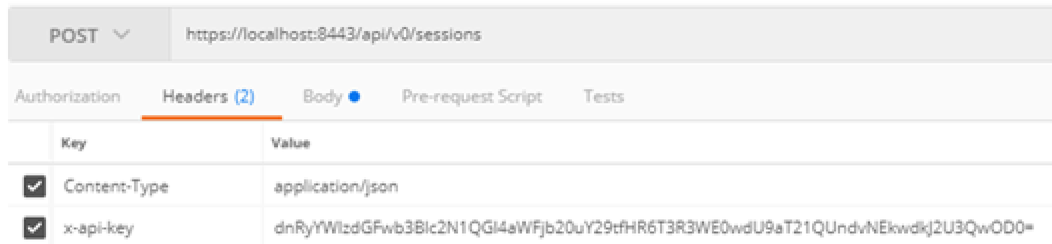
To enable or disable authentication on Linux, run the `configRestAuth.sh` script in `/opt/ixia/ixloadgateway` with the following switches:

`bash configRestAuth.sh --um-server 10.36.0.2`	Enable authentication and set the address of the User Management server
`bash configRestAuth.sh --disable-auth`	Disable authentication
`bash configRestAuth.sh --help`	List the available options

After enabling authentication on Linux, using REST requests on Linux is the same as on Windows.

Authenticating REST requests

An api-key is present in the request headers. This is 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"
}
```

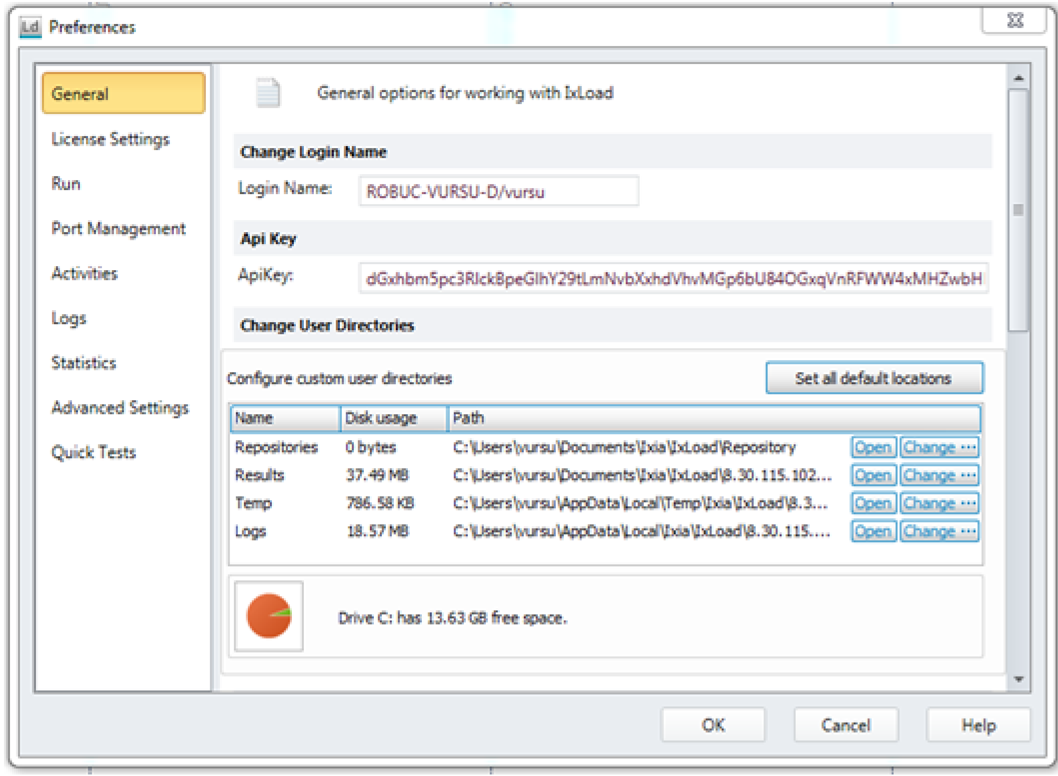
Users can delete only their own sessions (sessions that were created with the same api-key as the one provided during the DELETE request).

Retrieving the api-key

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

When authentication is turned on and you log on to Ixload with your Ixia User Management credentials, you can retrieve the `api-key` value from the General section of the Preferences widow (**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 copy the value of the `api-key` but you cannot modify it.

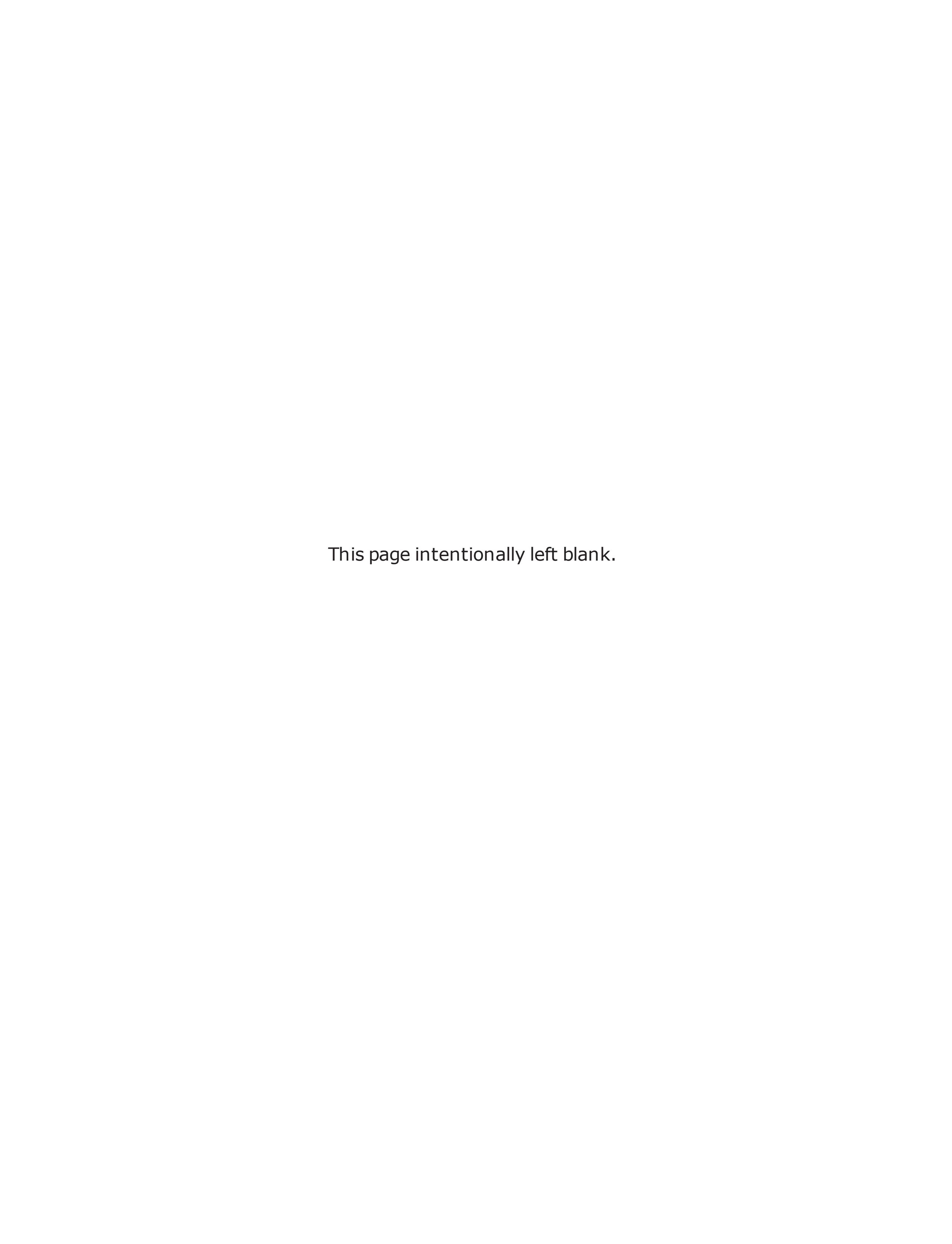


Script changes required for authentication

The changes that need to 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.



This page intentionally left blank.

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.

Other 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.

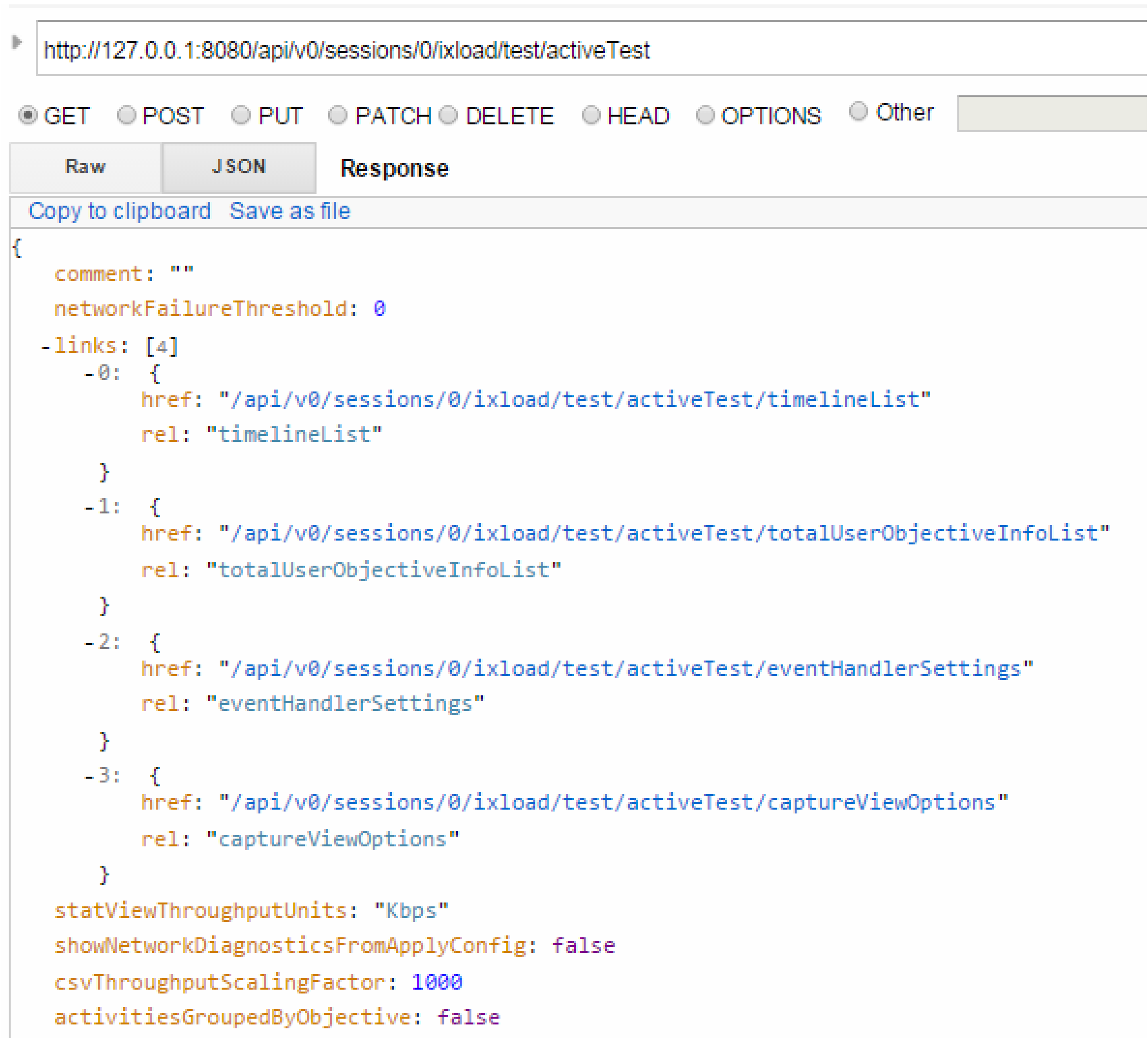
GET

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:

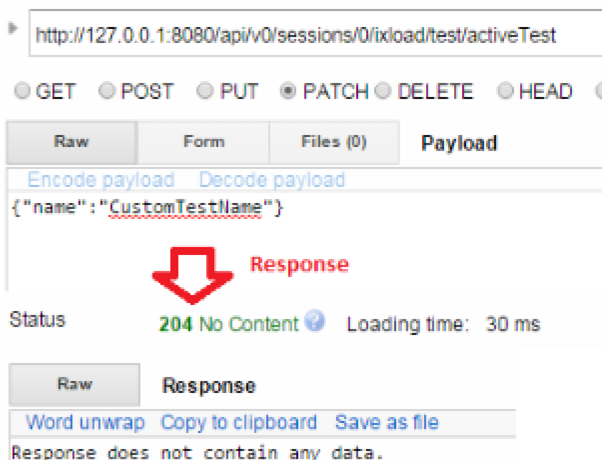


The preceding representation is only a demonstration of how the output of GET is provided in IxLoad REST API through the use of the Advanced REST Client in Google Chrome. The actual representation is different according to the programming language used to access the IxLoad REST API.

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.

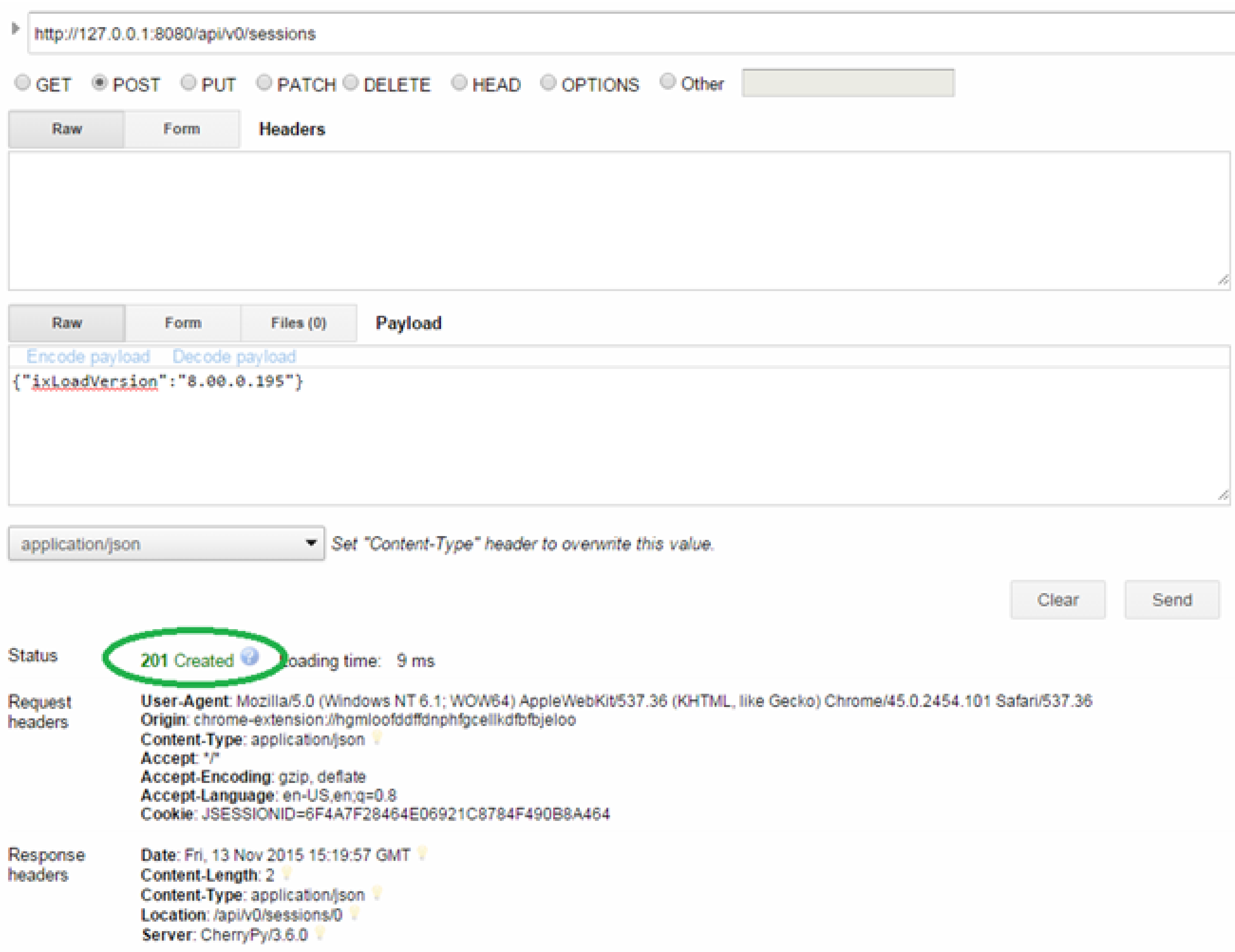
```
{
status: "PATCH operation failed"
error: "Cannot change HTTPClient1 at this moment. Please try again later”
}
```

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 emtpy ({}).

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



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:

http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/eventHandlerSettings

GET POST PUT PATCH DELETE HEAD OPTIONS Othe

Status 200 OK Loading time: 23 ms

Raw JSON Response

Copy to clipboard Save as file


```
{
  -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
      }
   }
}
```

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.

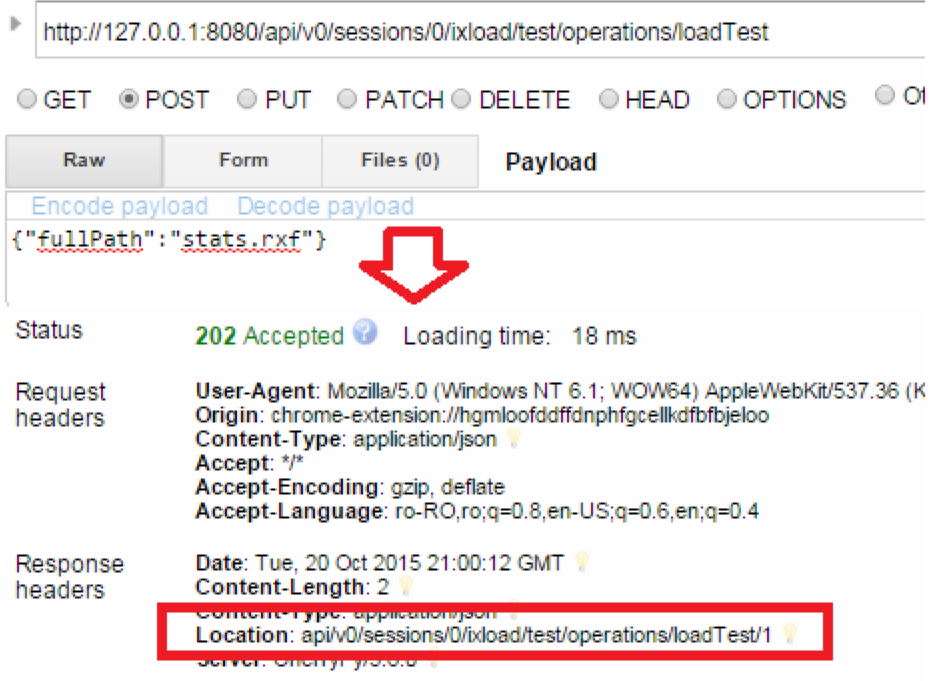
To check when operations are available for a certain resource, perform a GET request on the resource URL, adding '/operations' at the end of the URL. The following figure shows how the operations available for the test REST resource are represented:

The 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

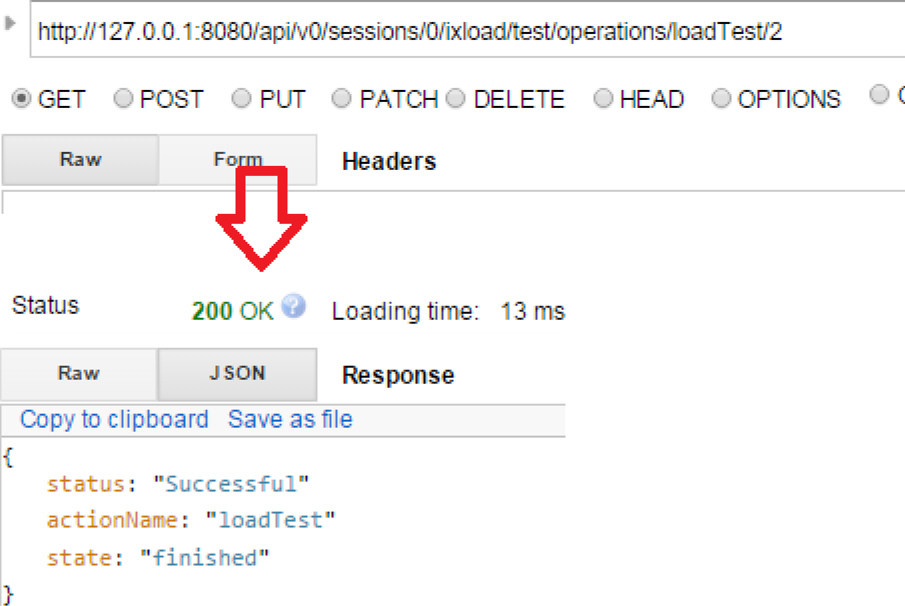
Starting an operation is done by performing a POST request on the following URL: $resourceUrl/operations/$operationName. The request payload represents the parameters required by

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:



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:



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:

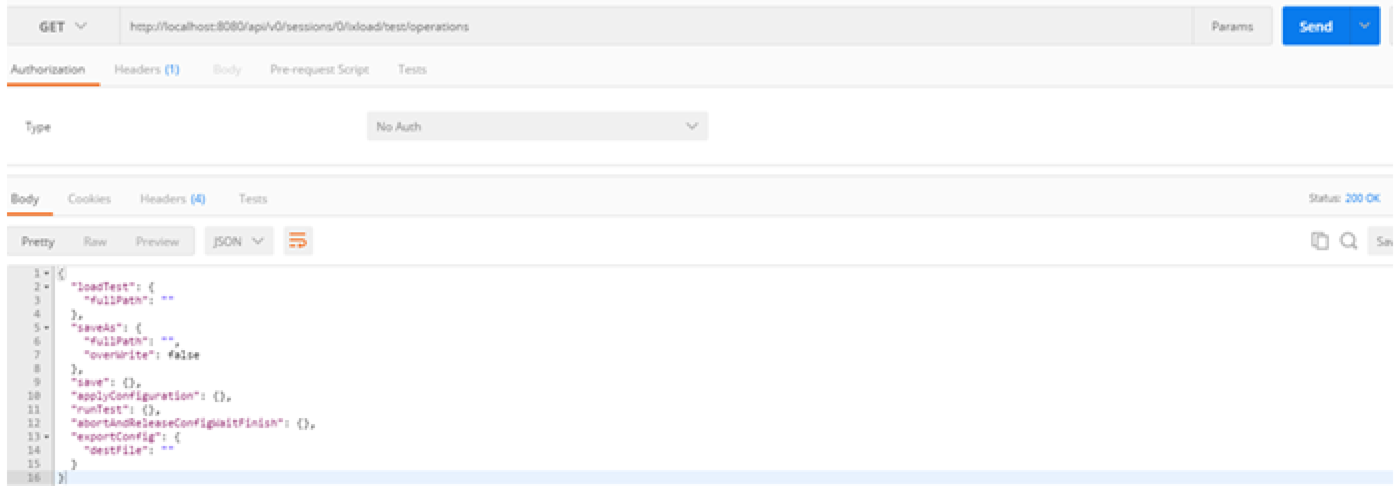
```
{
status: "Error"
actionName: "loadTest"
state: "finished"
error: "File doesn't exist - F:\statsdfs.rxf"
}
```

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 selecting **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 selecting **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.



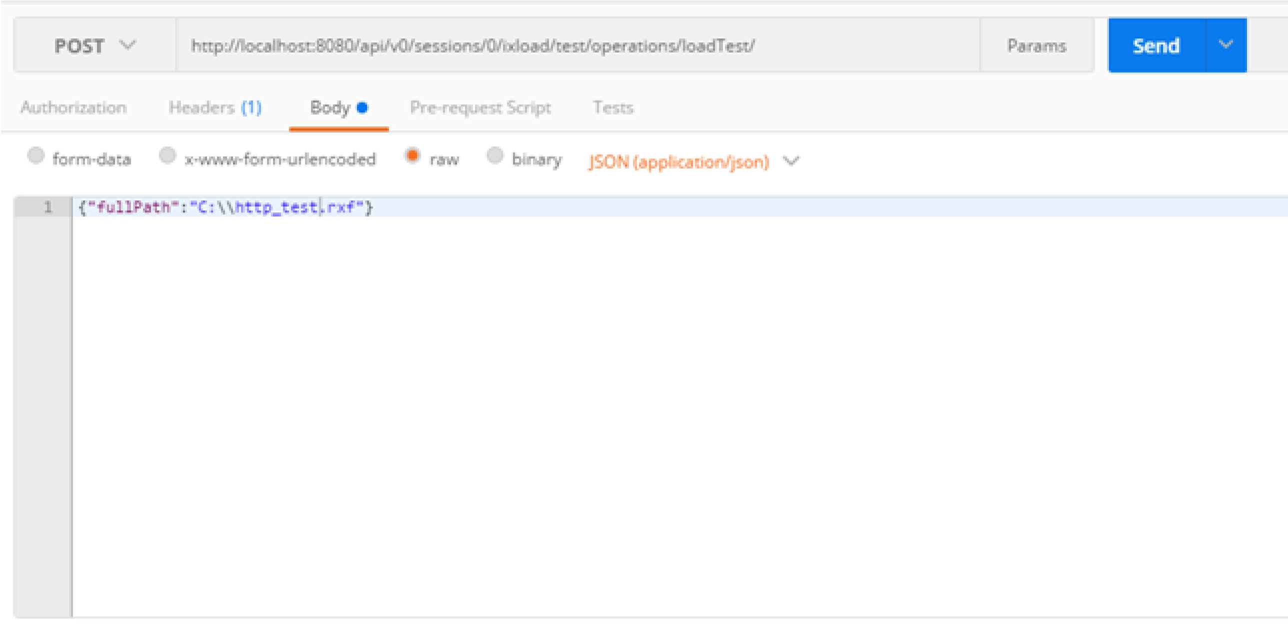
Example for loading a repository (.rxf) file

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

```
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
```

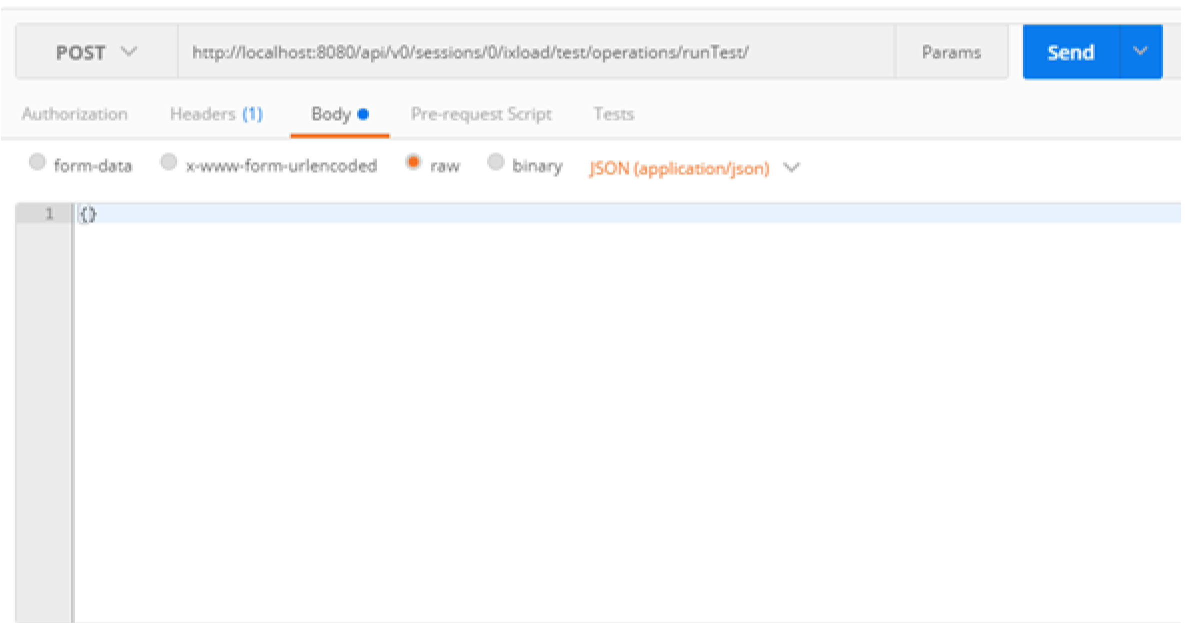
In the payload and body of the request, add the path to the .rxf file:

```
{"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

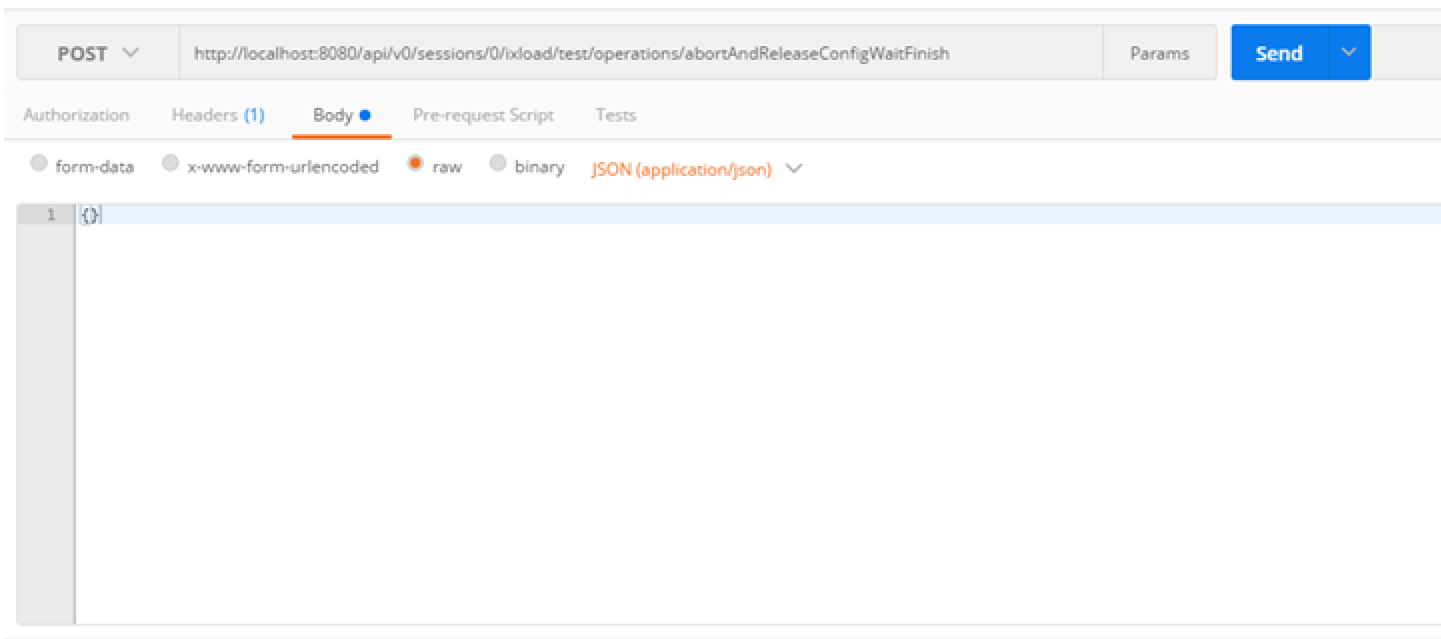
On an already created and activated session, do a POST on an URL similar to the following:

```
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
```

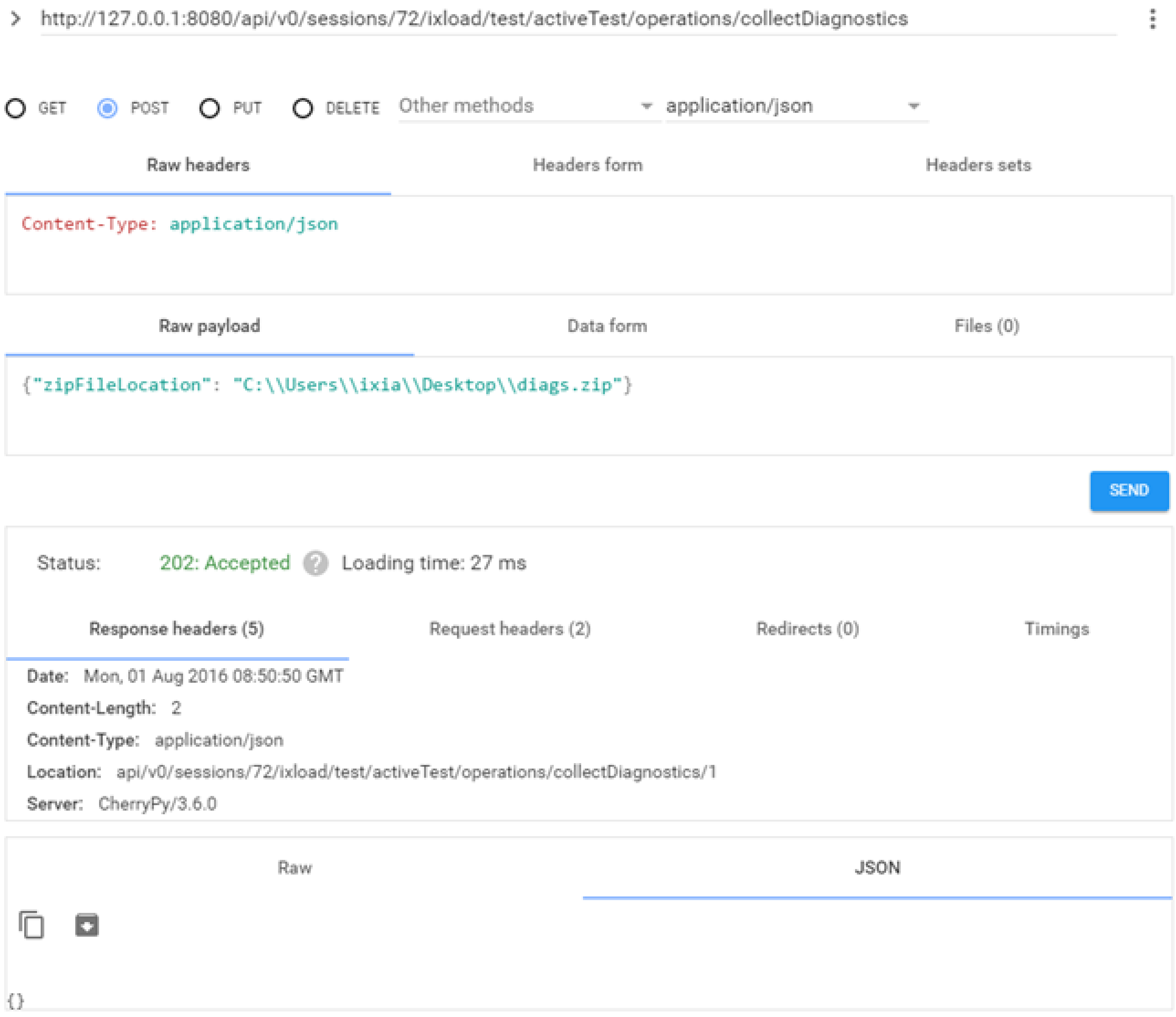
Specify the ZIP file location as the POST payload:

```
{"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:



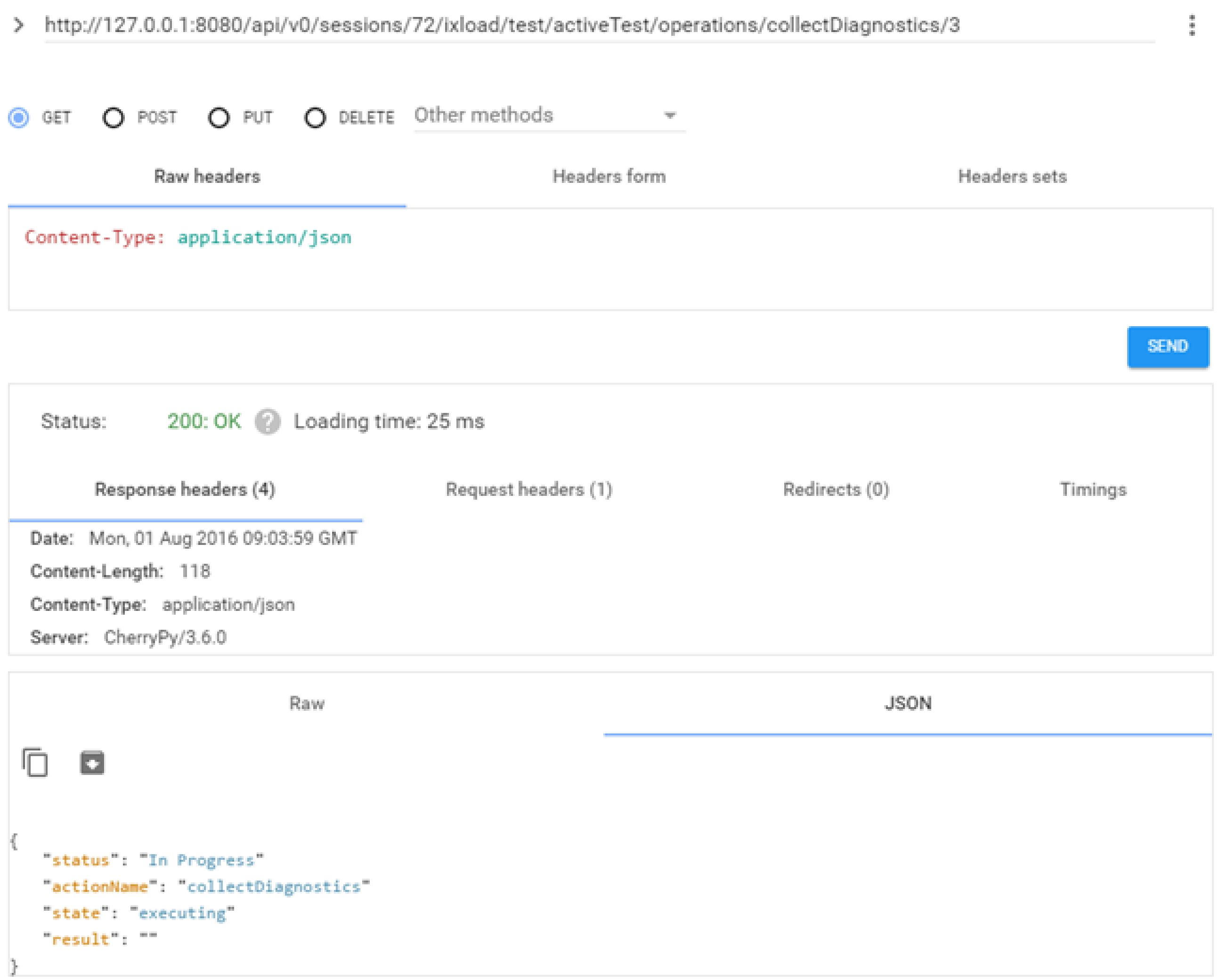
The status of the POST operation to collect diagnostics should be **202:Accepted**. The response to the operation should include a location.

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/collectD
iagnostics/1
```

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



Deleting the results directory after running a test

You can delete the results directory after running a test. This operation is available on the `test` resource, and requires the following:

- The request to delete the results directory must be made on the same session used to run the test that created the results directory.
- Only the results directory for the most recent test can be deleted.
- You did not unload the repository or load another repository after running the test.

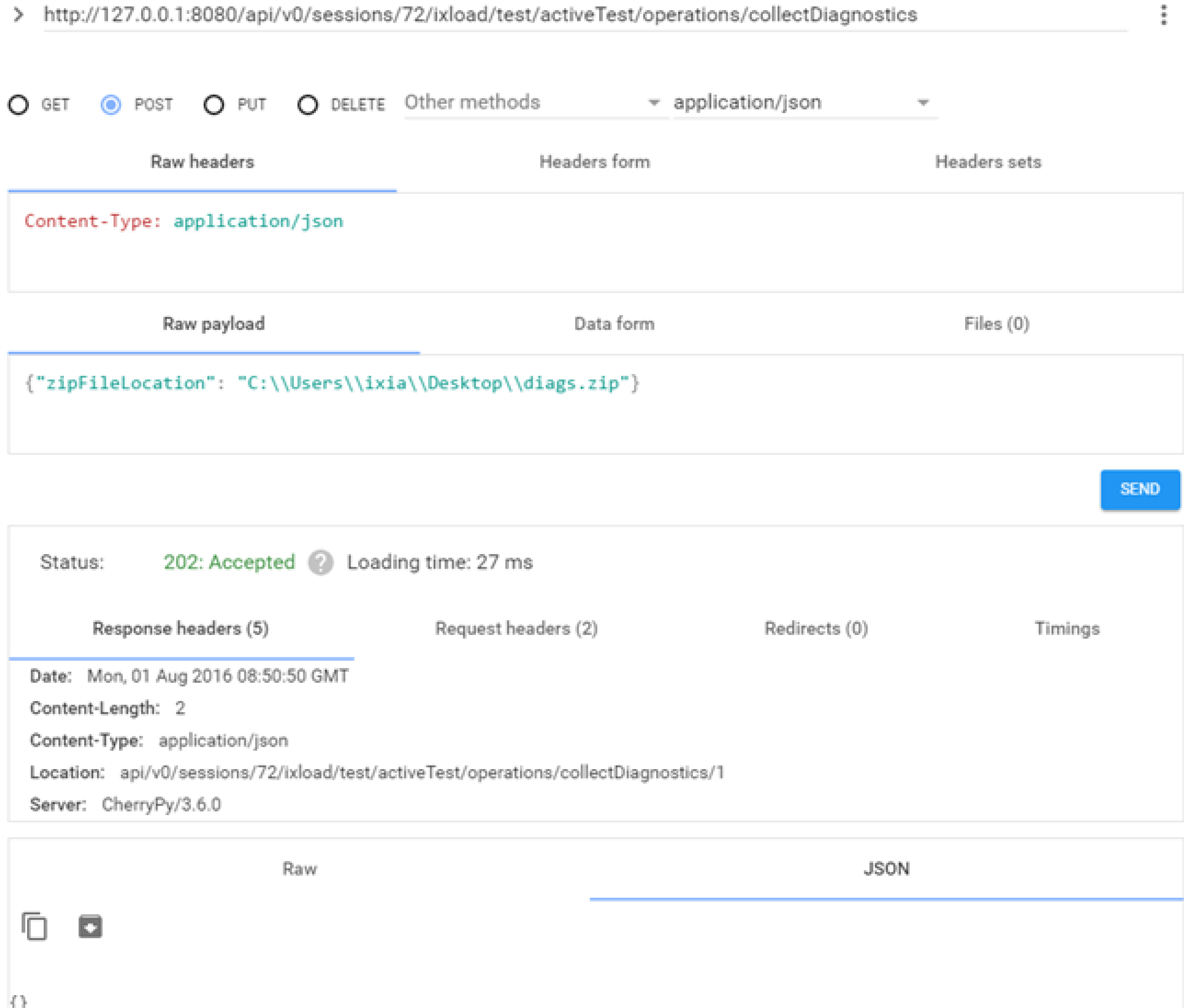
To delete the test result directory, use the following command:

```
POST @ api/v0/sessions/72/ixload/test/operations/deleteTestResultDirectory
```

This request does not require any parameters, so the request body should be empty: `{}`

This operation is useful for ensuring that the machine disk does not fill up with results directories. This is especially important for the IxLoad Linux solution.

After each test run, an automation script can use the APIs that are available to download any files of interest (csv files, port captures, etc.) and then use this operation to delete the results directory before closing the IxLoad session.



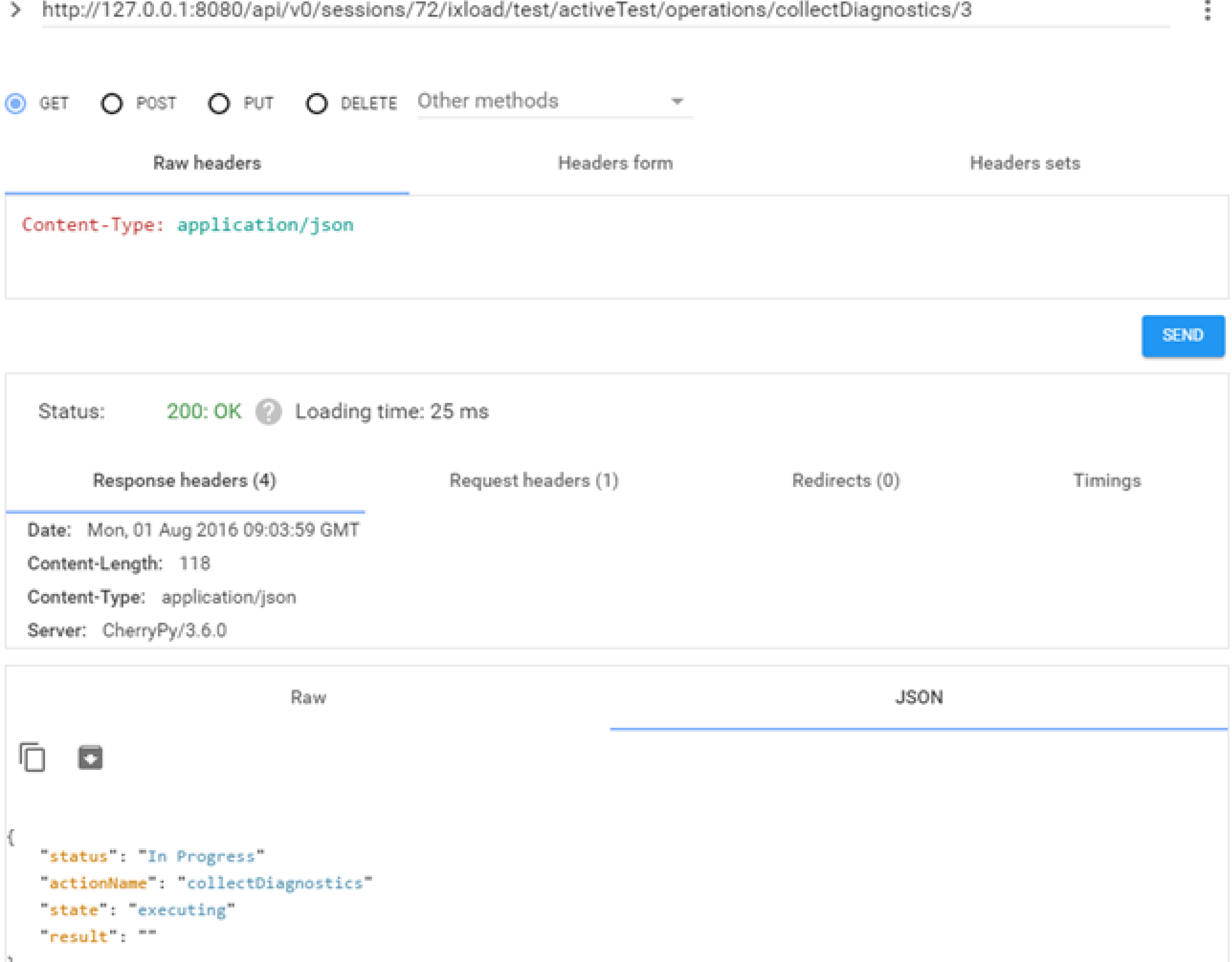
The status of the POST operation to collect diagnostics should be **202:Accepted**. The response to the operation should include a location.

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/collectD
iagnostics/1
```

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



extractDataModel operation

The `extractDataModelToFile` operation exports all the URLs that are available under the currently open IxLoad REST session. The operation is available to be executed (using a POST request) on the URL listed below, and outputs the information to a file on the disk. The path to the file is sent in the body of the operation (for example: : {"fullPath" : "D:/file.txt"}.

```
GET  https://localhost:8443/api/v0/sessions/0/ixload/operations
```

```
{
    "extractDataModelToFile": {
        "fullPath": ""
    }
}
```

The output file contains all the available URLs, and for each URL it specifies the options available under it, their current values in the configuration, and whether an option is read-only or not. Below are examples of how the `test` and `ipRange` resources appear in the output file.

Output for `test` resource:

```
80            - readOnly: False
81
82  Current resource: /ixload/test
83  Primitives:
84        - expirationTimer
85            - value:
86            - readOnly: False
87        - logs
88            - value: []
89            - readOnly: False
90        - loadedRxf
91            - value:
92            - readOnly: True
93        - runResultDirFull
94            - value: C:\ProgramData\Ixia\IxLoad\8.50.115.108\Results
95            - readOnly: False
96        - restObjectType
97            - value: ixTestController
98            - readOnly: True
99        - outputDir
00            - value: False
01            - readOnly: False
02
03  Current resource: /ixload/sessionOverview
04  Primitives:
```

Output for `iprange` resource:

```
Current resource: /ixload/test/activeTest/communityList/0/network/stack/childrenList/2/childrenList/3/rangeList/1
Primitives:
      - itemType
          - value: IpV4V6Range
          - readOnly: False
      - restObjectType
          - value: ixNetIpV4V6Range
          - readOnly: True
      - name
          - value: IP-R1
          - readOnly: False
      - count
          - value: 100
          - readOnly: False
      - ipType
          - value: IPv4
          - readOnly: False
      - ipAddress
          - value: 10.10.0.1
          - readOnly: False
      - enabled
          - value: True
          - readOnly: False
      - gatewayIncrement
          - value: 0.0.0.0
          - readOnly: False
      - randomizeSeed
          - value: 3029468524
          - readOnly: True
      - gatewayAddress
          - value: 0.0.0.0
          - readOnly: False
```

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

There two ways to create a session:

- By specifying the version of IxLoad to use to create the session.

  To specify the version to use, perform a POST on `/sessions` with one of the following payloads, appropriate for the URL type you are using:
  - /api/v0 URLs: `{“ixLoadVersion”:”8.XX.XX.XXX”}`
  - /api/v1 URLs: `{“applicationVersion”:”8.XX.XX.XXX”}`

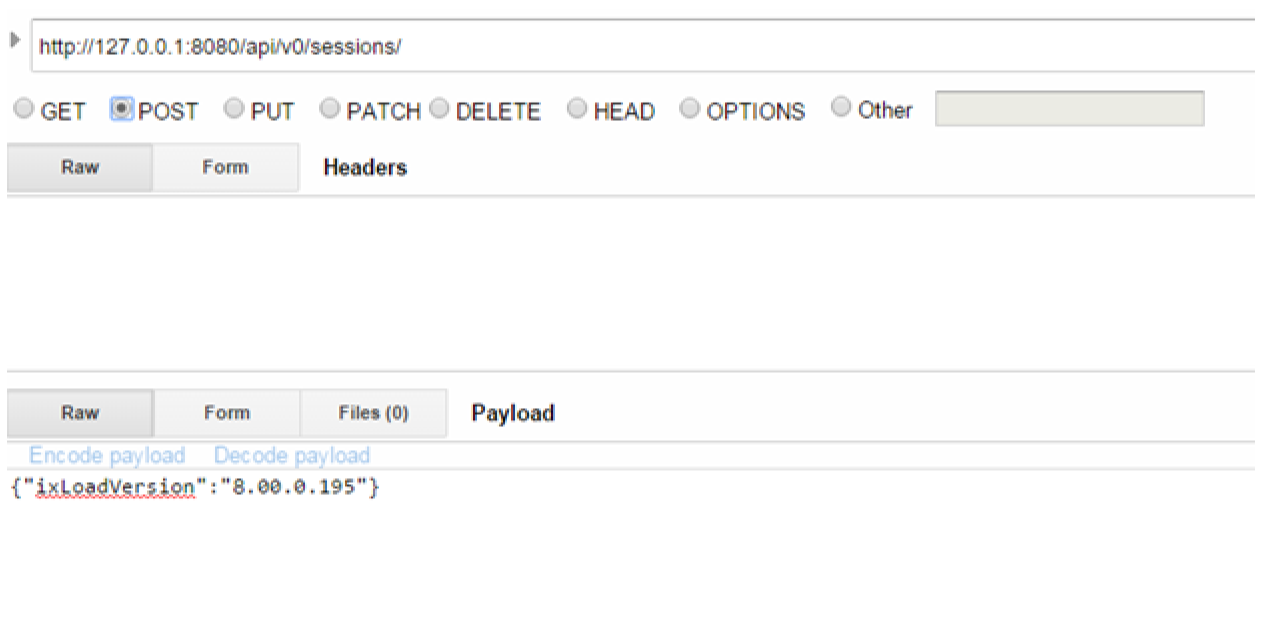
  then perform another POST on `/sessions/X/operations/start` to start the session.
- By automatically using the latest (or only) version installed to create the session

  `StartNewSession` creates a session using the latest version installed and then automatically starts it.

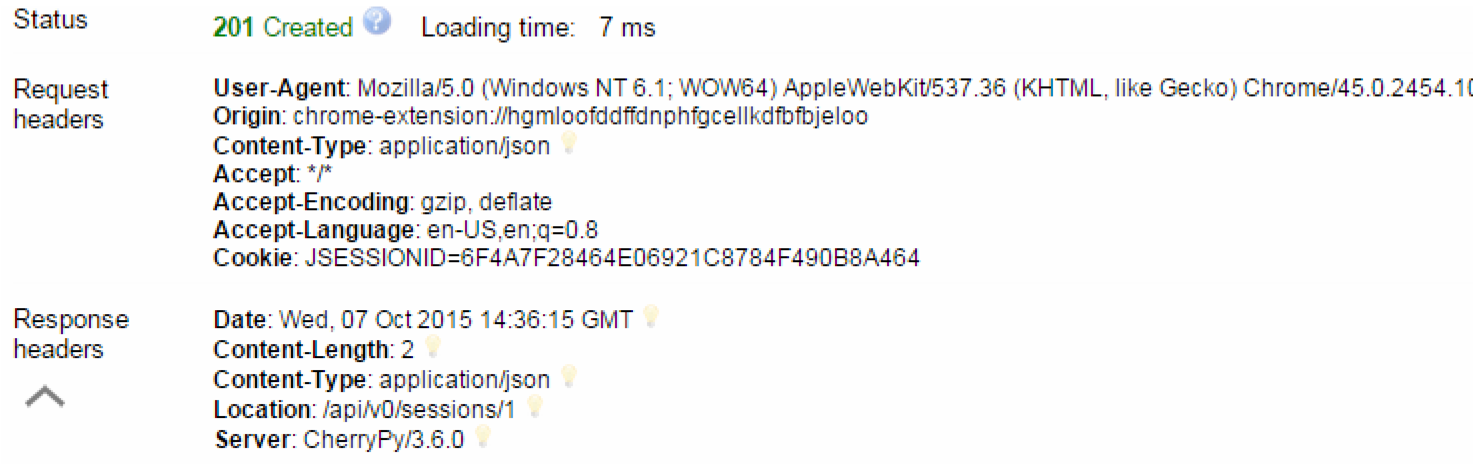
New session with a specified version

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:

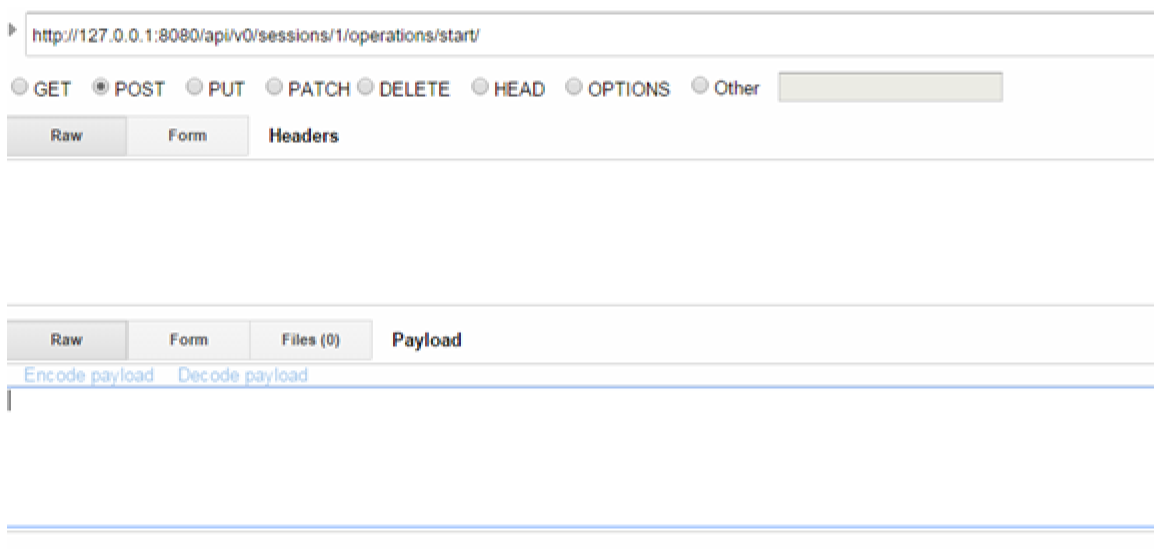


The following figure shows the response for the preceding post. Status is **201 Created** and the location points to the session.

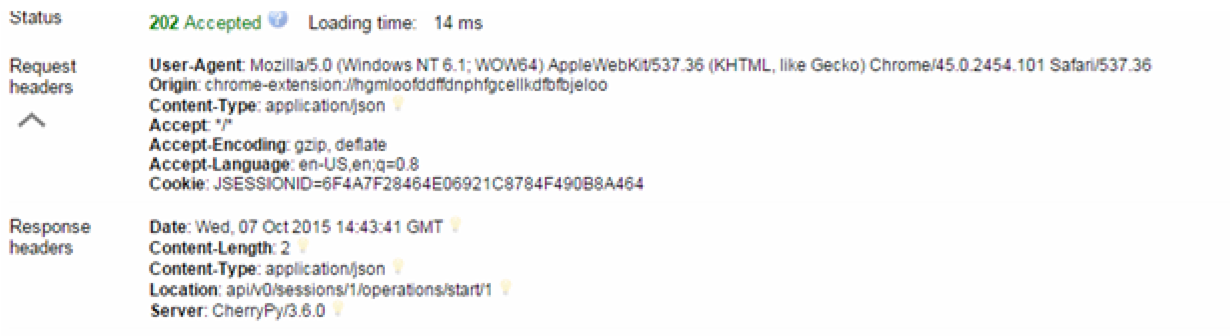


Starting a session

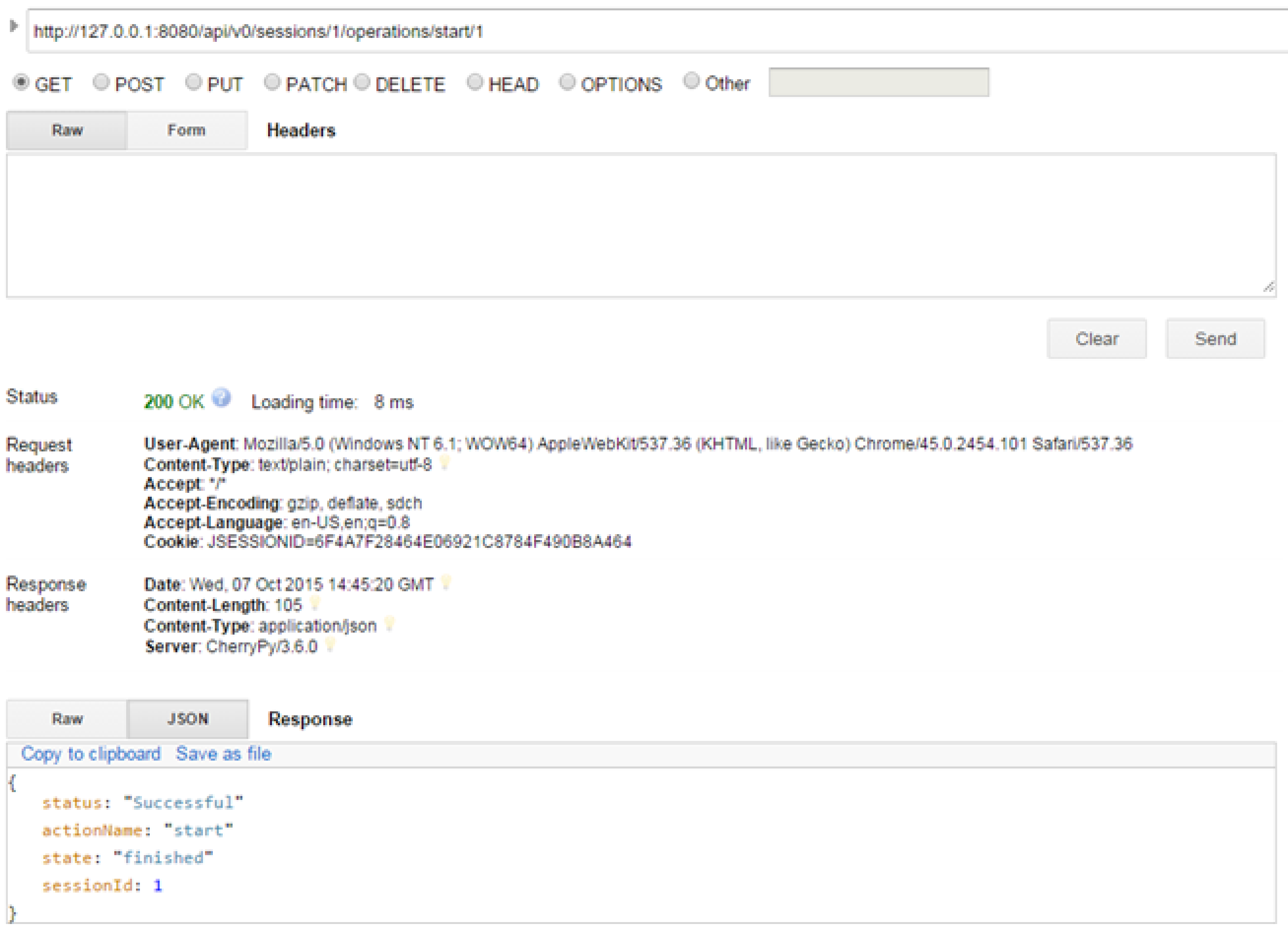
To start a specfied-version session, an operation is provided, named start. This operation is available on each individual session and requires no payload. The following figure shows how this looks in the REST client. This operation starts a new IxLoad session based on the IxLoad version for which the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



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.



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

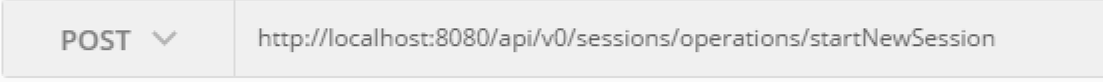
Raw JSON **Response**

Copy to clipboard Save as file

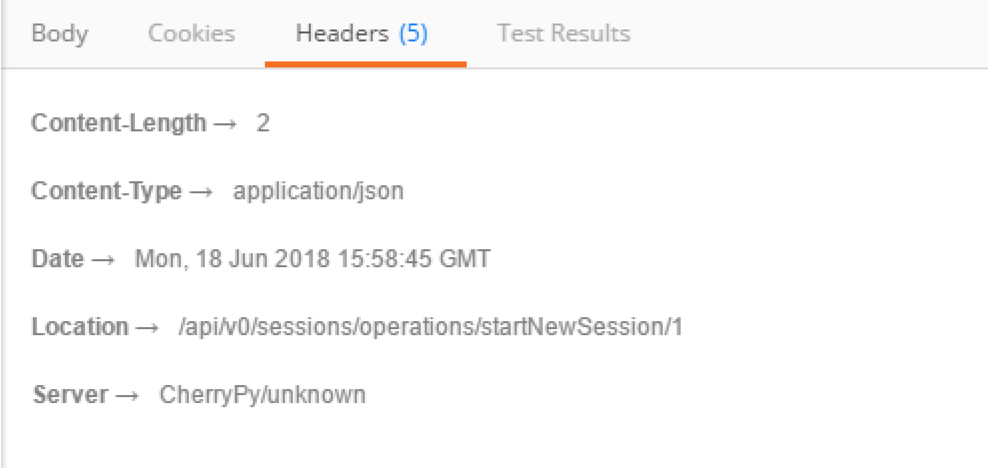

```
{
    status: "Successful"
    actionName: "start"
    state: "finished"
    errorMessage: "Already running maximum allowed copies of IxLoad."
    sessionId: 2
}
```


New session with the latest version

If you have only one IxLoad version installed, or you always want to use the latest installed version, you can create and start a new IxLoad session with a single POST request using the following URL:



`startNewSession` does not require a payload. As for every REST API operation, in the headers of the response there will be a location where the customer can check the status of the `startNewSession` operation:



After the operation finishes at the status URL there will be the application version that was used, along with the ID of the session that was created.

Pretty Raw Preview JSON

```
{
    "status": "Successful",
    "actionName": "startNewSession",
    "applicationVersion": "8.50.0.298",
    "sessionId": 0,
    "state": "finished",
    "result": ""
}
```

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.

Uploading and downloading files

You can upload and download files to and from the machine where IxLoadGateway is running.

Uploading files

Files can be uploaded to the machine where IxLoadGateway is running using the `resources` URL. To upload a file remotely, make a POST request in the following format:

```
https://10.114.198.17:8443/api/v0/resources?uploadPath=/mnt/ixload-
share/UploadRepository/demo.rxf&overwrite=true
```

The `uploadPath` parameter must be the absolute path where the file will be uploaded.

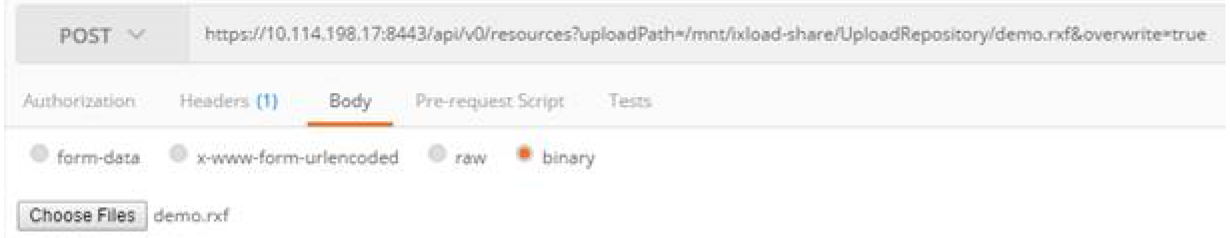
The `overwrite` parameter specifies if an existing file should be overwritten.

The POST request should contain the `Content-Type` header set to `multipart/form-data`:

The body of the POST request should represent the content of the file that will be uploaded, in binary format. From a script, this file can be sent as follows:

```
with open(fileName, 'rb') as f:
       headers = {'Content-Type': 'multipart/form-data'}
       params = {"overwrite": overwrite, "uploadPath": uploadPath}
       resp = requests.post(url, data=f, params=params, headers=headers, verify=False)
```

To upload a file from a tool like Postman, set the Body to `binary` and then choose the file to upload:



Downloading files

The `downloadResource` URL can be used to download files remotely from the machine where the IxLoadGateway service is running. Any file that the IxLoadGateway service can access can be downloaded.

To download a file, you perform a GET request on the following URL, where you will specify the IP of the machine where IxLoad Gateway is running, and the path to the file on that machine that you want to download.

```
https://IP:8443/api/v0/downloadResource?localPath=/mnt/ixload-share/file.rxf
```

This URL works for both Windows and Linux installations of IxLoadGateway, but you must specify the path in the correct Windows or Linux format.

API Browser

The API Browser enables you to view and modify the contents of an open IxLoad REST API session.

The tool is available on the root URL of the IxLoadGateway service:

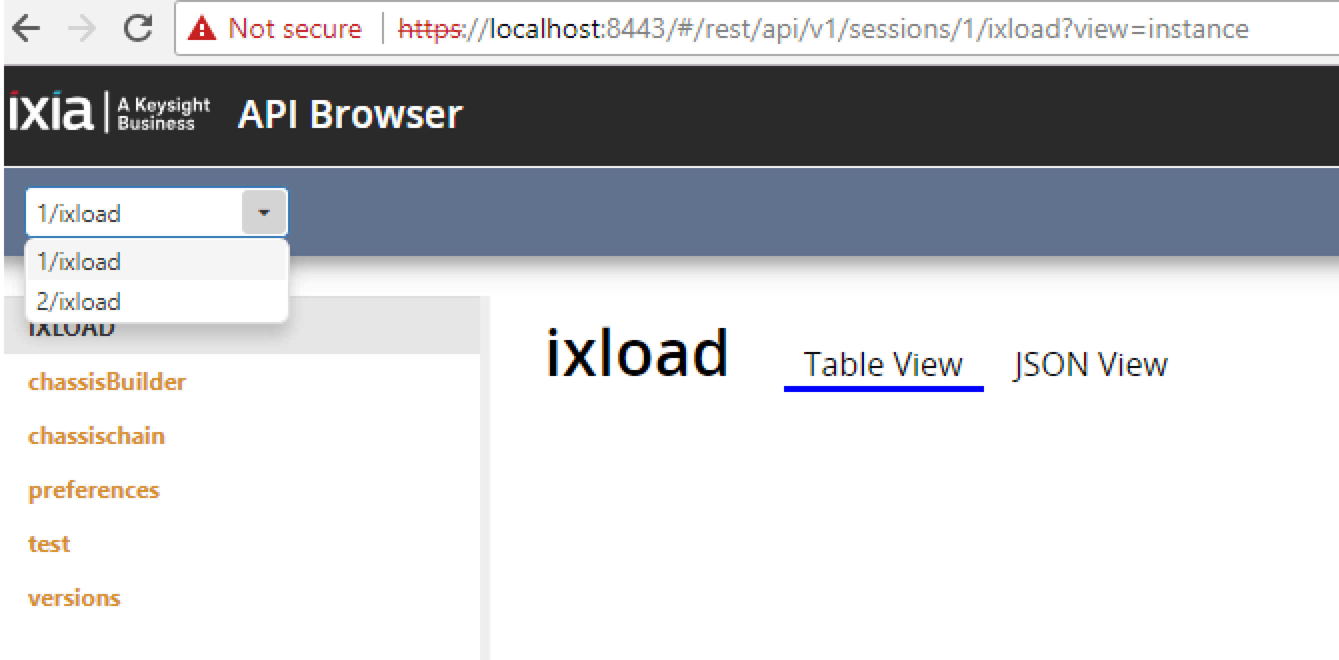
```
https://localhost:8443/
```

and

```
http://localhost:8080/
```

Displaying an IxLoad REST Session

To view the content of an IxLoad REST session, you select the desired session from the upper-left corner of the API Browser page:



After selecting a session, the data model can be viewed by selecting nodes in the tree on the left of the page. In the API Browser, you can:

View the data model in a tree structure:

ixia | A Keysight Business **API Browser**

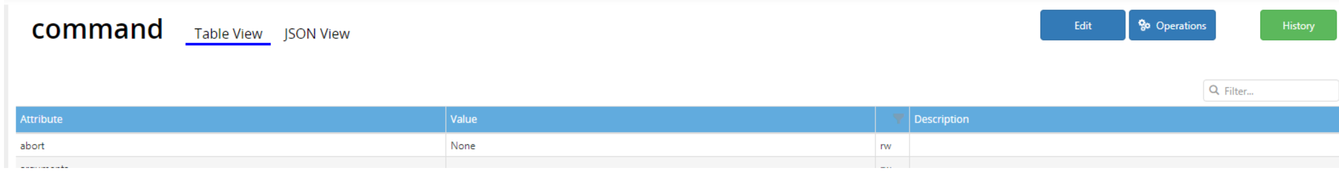
1/ixload ▾ | /test | /activeTest | /communityList | /0 | /activityList | /0 | /agent | /actionList | /2

- IXLOAD
- chassisBuilder
- chassischain
- preferences
- test
  - activeTest
    - activityLinks (empty)
    - captureViewOptions
    - communityList
      - communityList/0
        - activityList
          - activityList/0
            - agent
              - actionList (count = 3)
                - actionList/2
              - authProfileList (empty)
              - cmdPercentagePool
              - headerList (count = 4)
              - ipMappingList (empty)
              - methodProfileList (empty)
              - profileList (empty)
              - sslProfileList (empty)
          - network
          - networkActivityList (empty)
          - traffic
    - eventHandlerSettings
    - sessionSpecificDataList (count = 2)
    - timelineList
    - totalUserObjectiveInfoList (empty)
  - logs (empty)

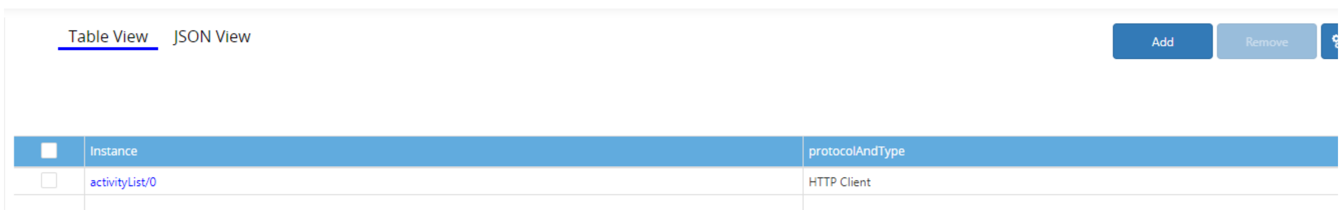
command Table View JSON View

Attribute	Value
abort	None
arguments	
cmdName	Get 1
commandType	GET
destination	None
enableDi	0
method	-1
namevalueargs	
pageObject	None
pingFreq	10
profile	-1
restObjectType	ixHttpCommand
sendingChunkSize	None
sendMD5ChkSumHeader	0
sslProfile	-1
streamIden	3
useSsl	0
windowSize	65536

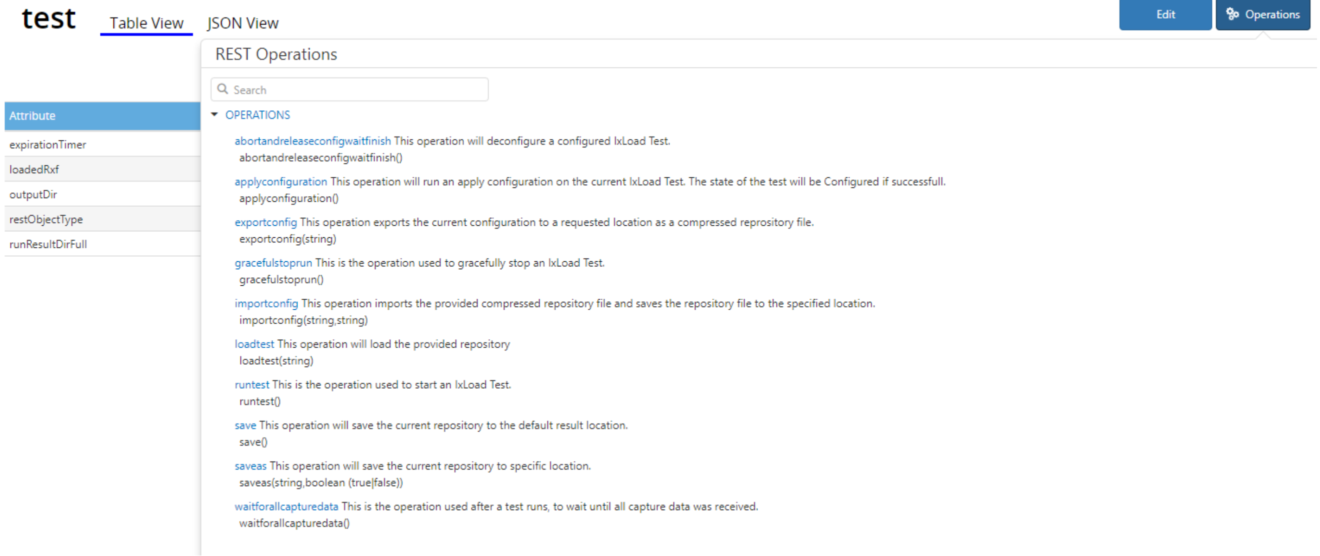
Edit field values, using the Edit button. This can be used to modify primitive values (numbers, strings, Booleans) for all fields that are not read-only.



Add or remove elements from lists, using the Add and Remove buttons:



Execute async operations, using the Operations button. This section contains all the actions available under `/resourceUrl/operations` in the REST API.



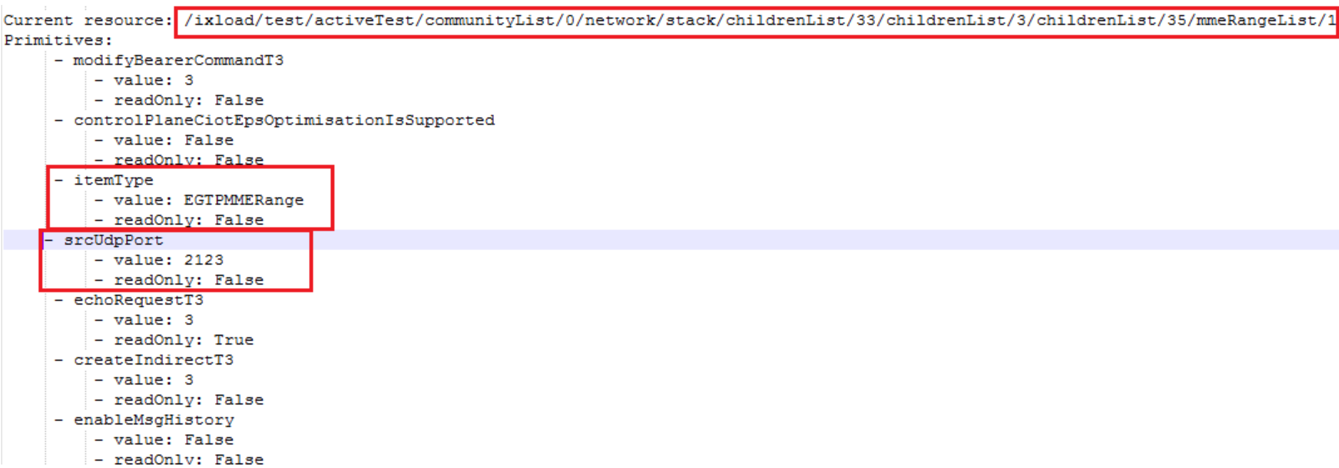
How to find URLs in a REST API session

The IxLoad data model is very large, and it can be difficult to find the REST API option that corresponds to an option in the IxLoad GUI, either from scripts, the API Browser, or a tool such as Postman.

You can use the `extractDataModelToFile` operation to find options. To do this, you load the repository in a REST session, and then use `extractDataModelToFile` to export all the available URLs to a file on the disk.

For example, assume you want to find the `sourceUdpPort` inside an MME Range. In the GUI, `sourceUdpPort` is in the following location:

If you search for either `udpPort` or its value, `2123`, in the file created by `extractDataModelToFile`, you will find the following information in the file:

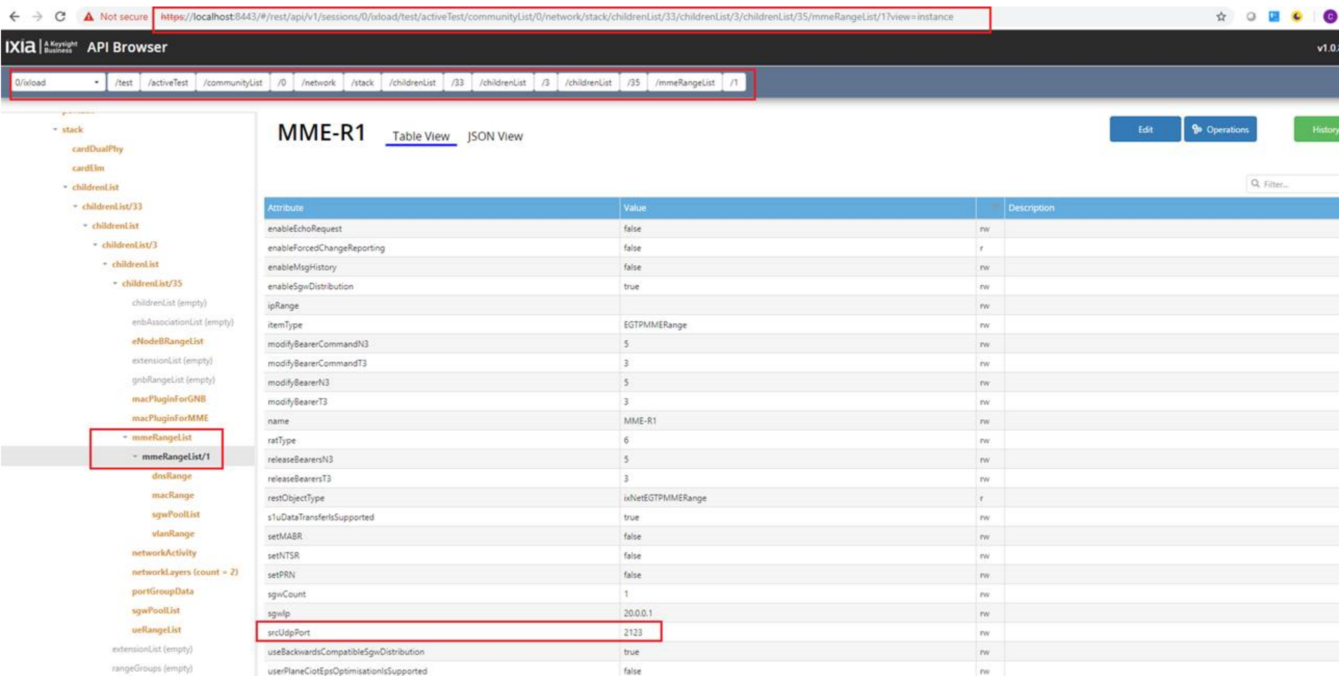


```
Current resource: /ixload/test/activeTest/communityList/0/network/stack/childrenList/33/childrenList/3/childrenList/35/mmeRangeList/1
Primitives:
    - modifyBearerCommandT3
        - value: 3
        - readOnly: False
    - controlPlaneCiotEpsOptimisationIsSupported
        - value: False
        - readOnly: False
    - itemType
        - value: EGTPMMERange
        - readOnly: False
    - srcUdpPort
        - value: 2123
        - readOnly: False
    - echoRequestT3
        - value: 3
        - readOnly: True
    - createIndirectT3
        - value: 3
        - readOnly: False
    - enableMsgHistory
        - value: False
        - readOnly: False
```

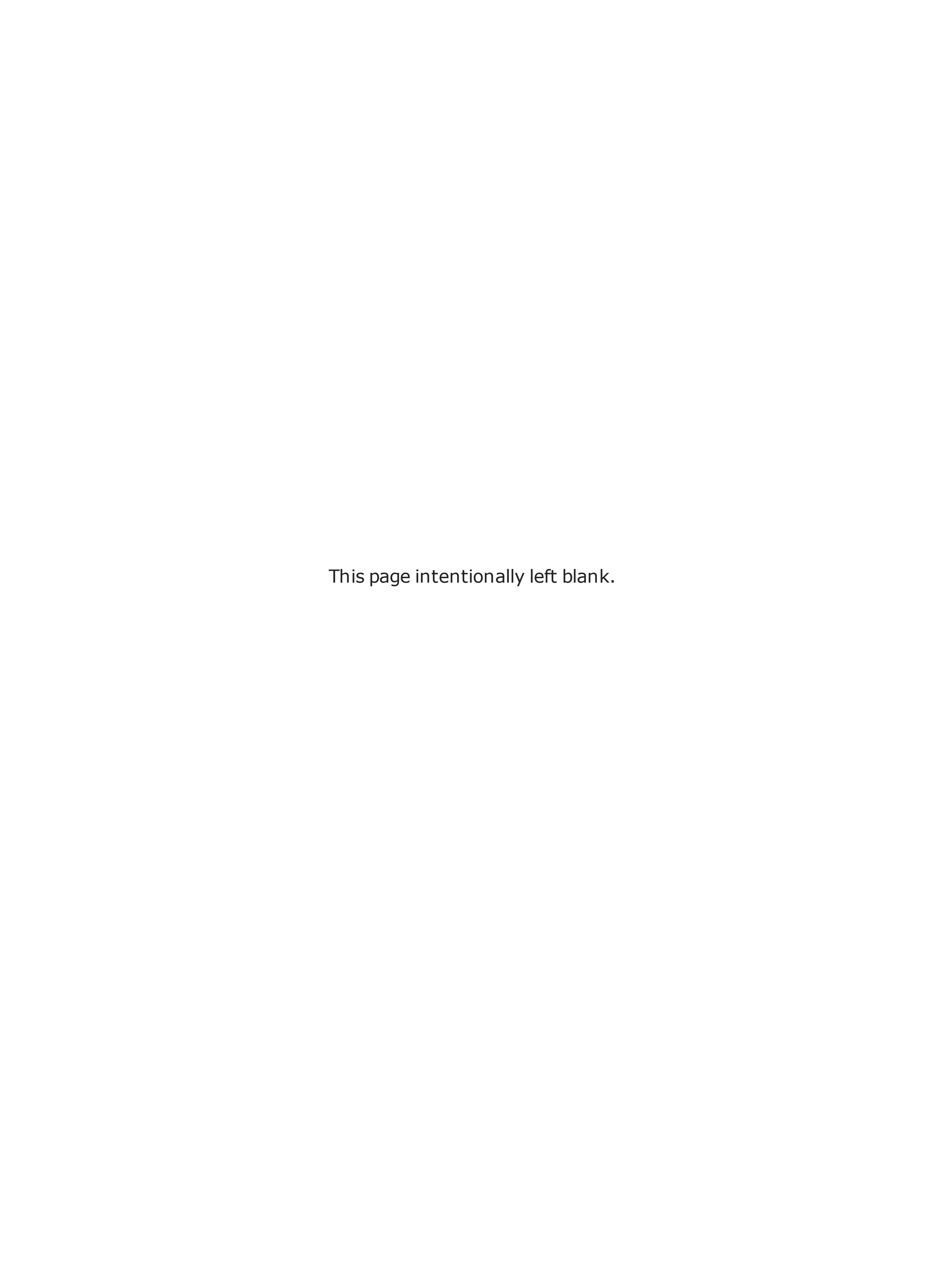
This shows the URL where the MME Range can be located in the REST session, and the name of the field inside the REST session. You can copy the URL in the API Browser (or in a script), in the following format:

`https://{IP}:8443/#/rest/api/v1/sessions/{sessionID}/` + URL retrieved from the file

For the MME Range example above, copying the URL in the API Browser shows the correct resource:



You can use this workflow to find any parameter in the REST API, keeping in mind that when searching in the exported data model file, a resource's label in the IxLoad GUI may not be identical to its name in the REST API. For example, the **Source UDP Port** option in the GUI is `srcUdpPort` in the REST API.


This page intentionally left blank.

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: L4-7 plugins, L2-3 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"

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
  - 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 L2-3 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" }

DUTs

You can find the list of DUTs (devices under test) on the following path:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/dutList/
```

```
[
    {
        "comment": "",
        "name": "DUT1",
        "objectID": 0,
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/test/activeTest/dutList/0/docs",
                "rel": "docs"
            },
            {
                "href": "/api/v0/sessions/0/ixload/test/activeTest/dutList/0/dutConfig",
                "rel": "dutConfig"
            }
        ],
        "restObjectType": "ixDut",
        "type": "Firewall",
        "scenarioElementType": "dut-basic"
    },
    {
        "comment": "",
        "name": "DUT2",
        "objectID": 1,
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/test/activeTest/dutList/1/docs",
                "rel": "docs"
            },
            {
                "href": "/api/v0/sessions/0/ixload/test/activeTest/dutList/1/dutConfig",
                "rel": "dutConfig"
            }
        ],
        "restObjectType": "ixDut",
        "type": "ServerLoadBalancer",
        "scenarioElementType": "dut-basic"
    },
```


IxLoad supports 5 types of DUT:

- Firewall
- Server Load Balancer (SLB)
- External Server
- Packet Switch
- Virtual DUT

You can choose to view only a specific type of DUT by performing a GET operation on the DUT list, and including a query string that specifies the DUT type.

For example, to view the list of firewall DUTs:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/dutList/?filter="type eq firewall"
```

To add a new DUT, perform a POST operation on the same list, specifying the type.

For example, to add a new firewall DUT:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/dutList/
{"type" : "Firewall"}
```

To delete a DUT, perform a DELETE at the following address:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/dutList/$
dutObjectID
```

To modify the properties of a DUT, use the PATCH operation.

DUT resources:

`dutConfig`: The configuration properties of the device (particular to that type of DUT) can be found in the following list:

```
http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/dutList/$
dutObjectID/dutConfig
```

Expiration timer

The expiration timer (`expirationTimer`) enables you to flag sessions for deletion after a fixed length of time has elapsed. This option is useful for preventing stalled automation scripts from keeping IxLoad REST sessions open infinitely.

If the timer expires and the session is in the Unconfigured state (that is, it was not running a test), then the session is immediately deleted.

If the timer expires and the session is in a state other than Unconfigured (for example, the Running state), then the session is first transition to the Unconfigured state, and then deleted.

This timer is exposed as a field named expirationTimer under a URL like the following:

```
http://<IP_ADDRESS>:8080/api/v0/sessions/{sessionId}/ixload/test/
```

To modify the value of this field, you execute an HTTP PATCH request on the URL. The expiration timer starts running when the PATCH request is executed.

The value formats for expirationTimer are as follows:

Format	Description
`1 day`	1 day
n `days`	n number of days
`hh:mm:ss`	hours:minutes:seconds

For example:

Value	Description
1:20:30	1 hour, 20 minutes and 30 seconds
2 days	2 days
2 days, 1:20:30	2 days, 1 hour, 20 minutes and 30 seconds

The expiration timer can be updated to a new value at any time. The update resets the expiration timer to a new value, meaning that the session will be deleted after the new timer has expired.

To cancel the expiration timer, execute a PATCH request with no value:

```
{“expirationTimer”: “”}
```

Enabling Analyzer and downloading captures

You can enable Analyzer and retrieve port captures from the IxLoad REST API.

To enable Analyzer on a port, execute a PATCH request on a URL of the form:

```
http://<IP_ADDRESS>:8080/api/v0/sessions/
{sessionId}/ixload/test/activeTest/communityList/{communityListId}/network/portList/
{portListId}
```

with the following payload:

```
{“enableCapture”: “True”}
```

Once capture is enabled on a port, a new URL will be available under the `portList`:

```
http://<IP_ADDRESS>:8080/api/v0/sessions/
{sessionId}/ixload/test/activeTest/communityList/{communityListId}/network/portList/
{portListId}/restCaptureFile
```

To download the capture file from the port after the test has finished running, execute a GET request on the URL.

To ensure that the captures are ready to be downloaded, you should call the `waitForAllCaptureData` operation after the test has finished running:

POST	http://localhost:8080/api/v0/sessions/1/ixload/test/operations/waitForAllCaptureData

`waitForAllCaptureData` does not require any payload, and will block until all capture files have been copied on the machine where the IxLoad client is running.

If the GET request is executed from a browser, then the browser will prompt for the location to download the capture to. If the GET request is performed from the IxLoad sample scripts, then you can provide the path where the captures should be downloaded to.

Ixia recommends downloading the captures either by using a browser or through scripts since UI REST clients can hang or crash if the captures are too large.

If you use a UI REST client such as Postman, the captures will be downloaded to the Results folder on both Windows and Linux.

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}
```

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:

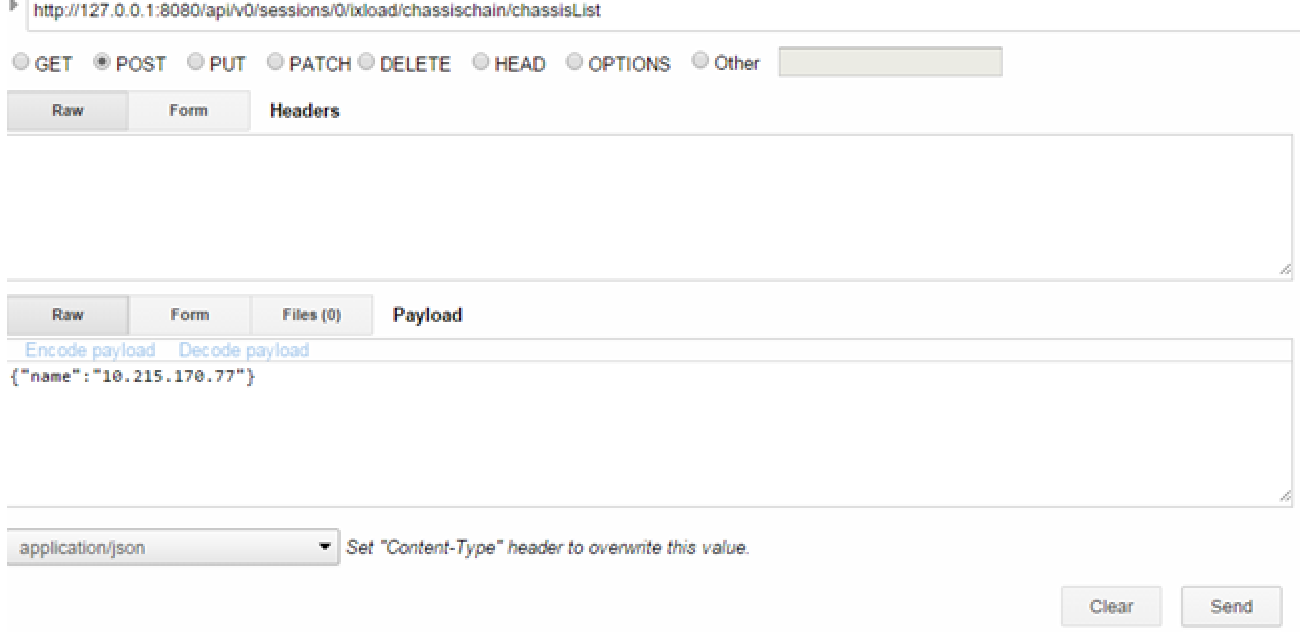
- http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList

Adding a chassis

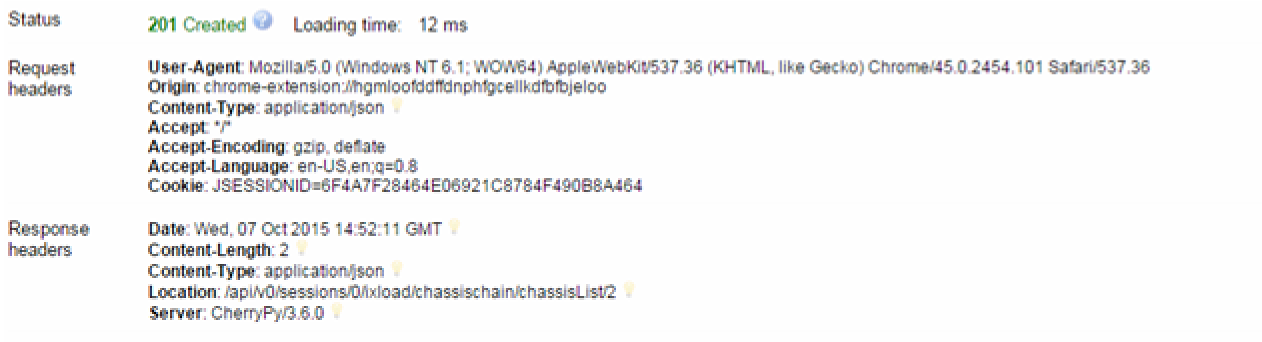
Add a chassis as follows:

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

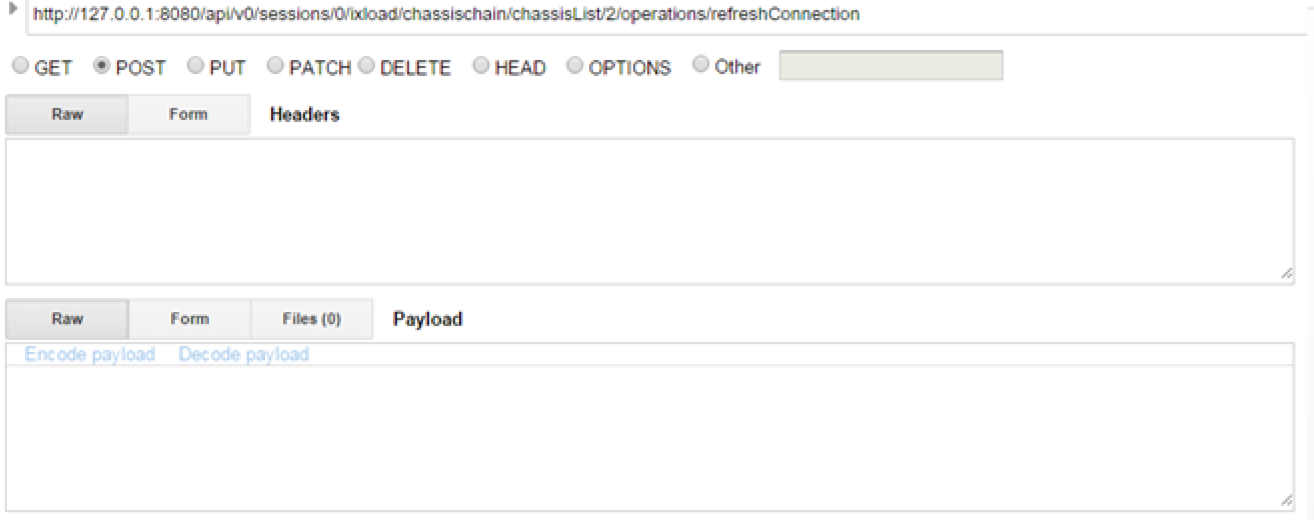


Connecting to a chassis

To connect to a chassis, do the following:

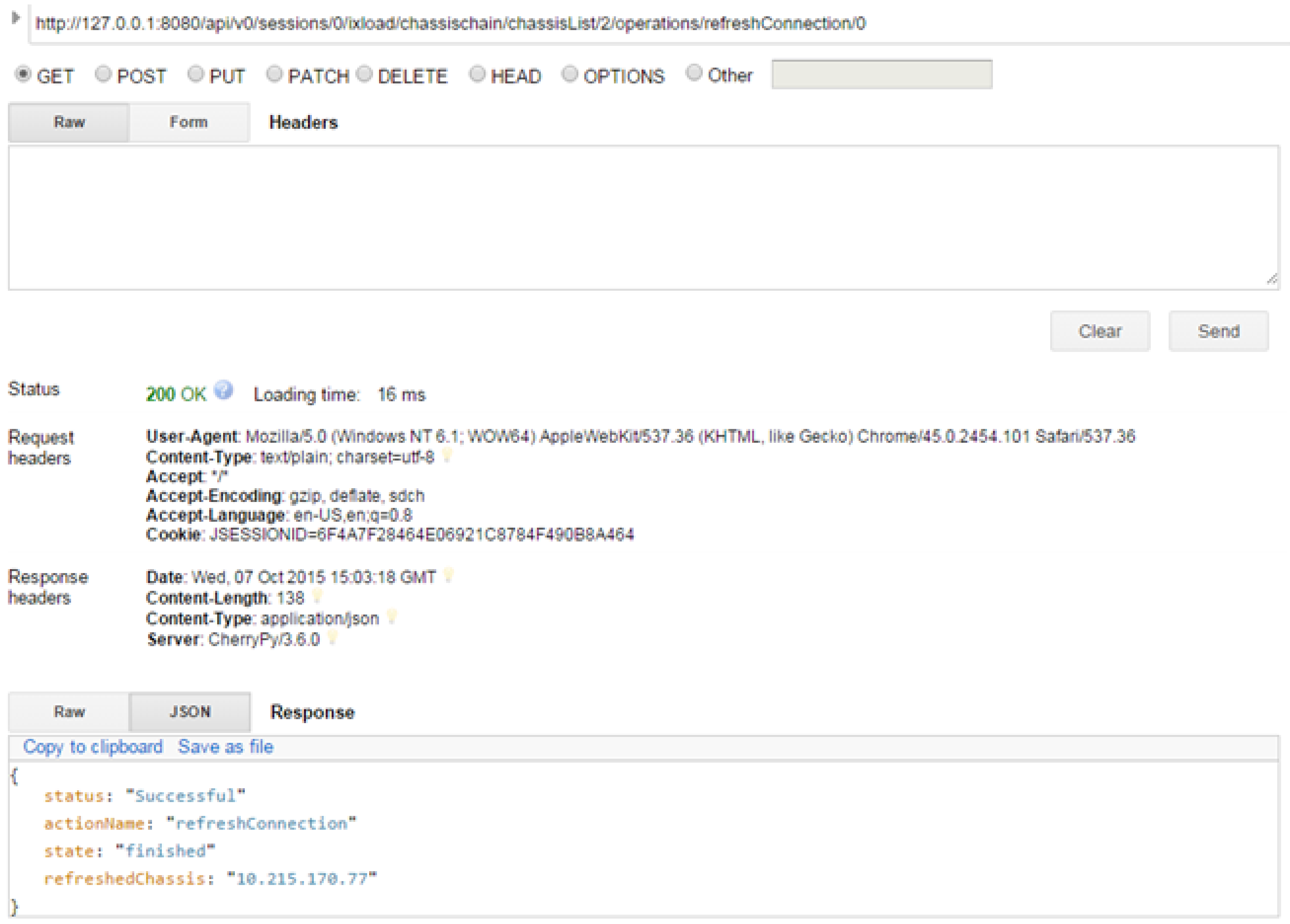
```
POST @
api/v0/sessions/0/ixload/chassischain/chassisList/2/operations/refreshConnection
```

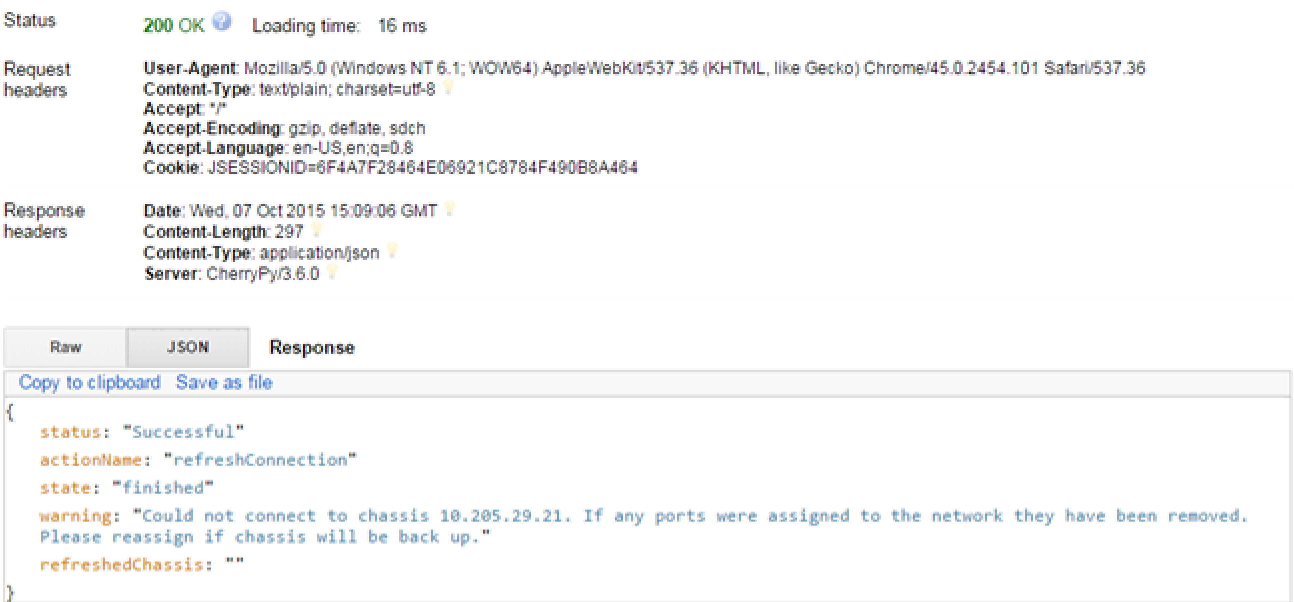
No payload is required. The following figure shows how this looks in the REST client:



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

The result to the refresh operation is as follows:





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.

Assigning 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

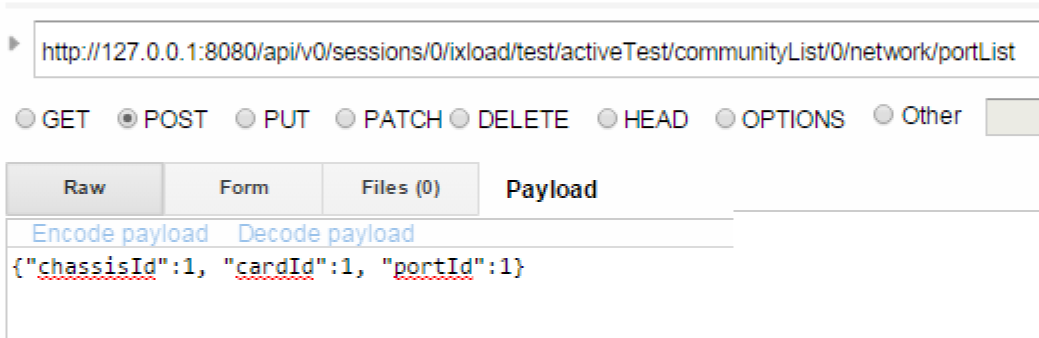
GET POST PUT PATCH DELETE HEAD OPTIONS Other

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/cardl
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:



Unassigning 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.

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
```

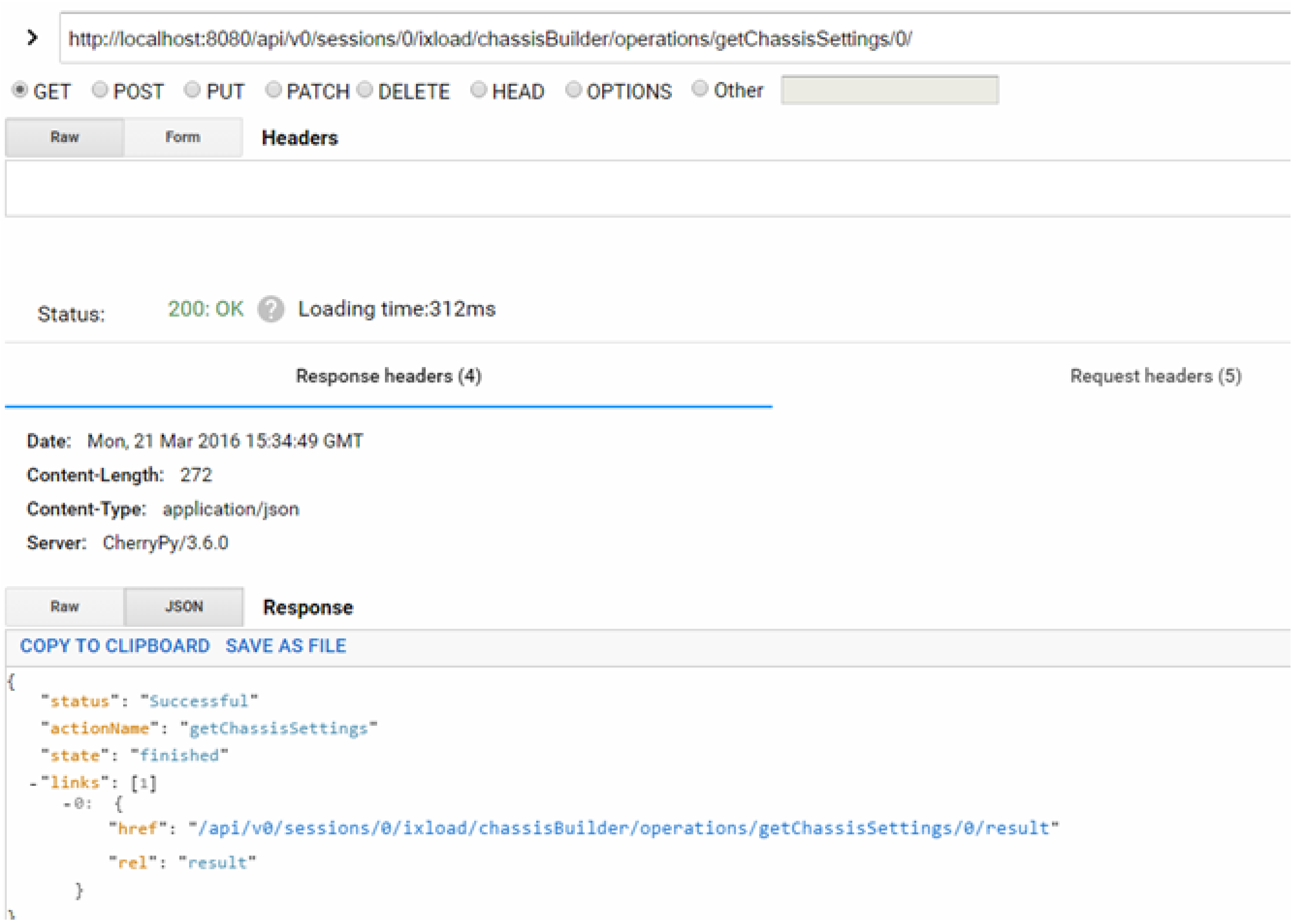
```
{
  -"deleteCard": {
      "cardId": ""
   }
  -"updateChassisSettings": {
      "enableLicenseCheck": null
      "ntpServer": null
      "licenseServer": null
      "txDelay": null
   }
   "getChassisSettings": {}
   "hardChassisReboot": {}
  -"getCardPorts": {
      "cardId": ""
   }
  -"updatePortById": {
      "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}
```



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"
}
```

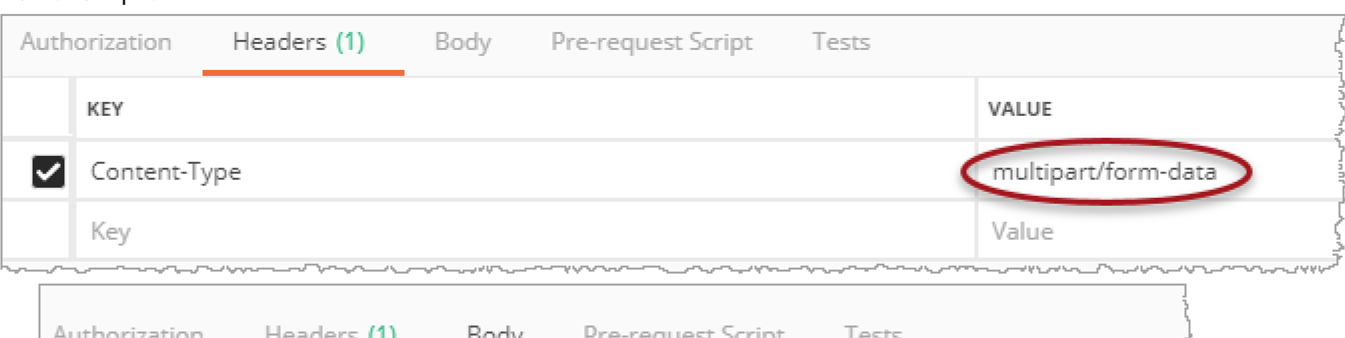
Upload and Download Diameter XML Configuration Files

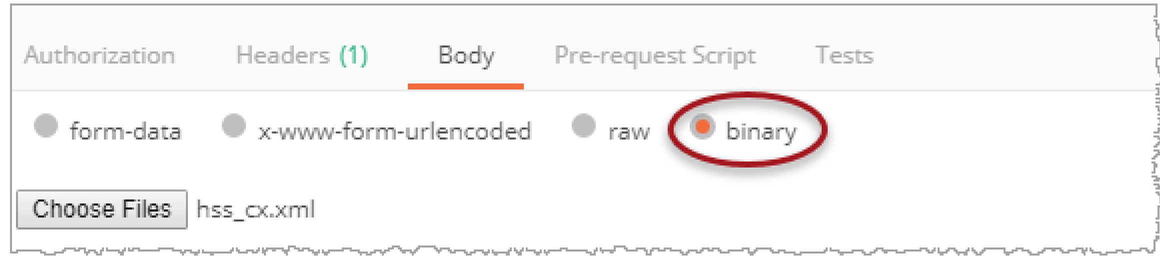
The IxLoad REST API provides support for uploading and downloading Diameter XML configuration files.

Upload

Assume that you have saved an IxLoad Diameter configuration file, named **hss_cx.xml**.

To upload the file, send a POST command with the following characteristics:

1. Insert the header **Content-Type: multipart/form-data**.
2. Attach as **Binary File** the **hss_cx.xml** config from the REST client.
   For example:
   

3. Send the entire command **POST**.
   For example:
   ```
   POST
   http://localhost:8080/api/v1/sessions/0/ixload/test/activeTest/communityList/0/network/stack/childrenList/2/childrenList/3/childrenList/4/DiameterPortgroupData/upload/?overwrite=true&uploadPath=D:/New%20Folder/Rest/Upload/testDPGD.xml
   ```
   The command can vary based on the test configuration.

The last part of the POST command (`D:/New%20Folder/Rest/Upload/testDPGD.xml`) identifies the location and name under which the Diameter configuration will be saved on the REST Gateway (`localhost`) and then imported in the rxf.

Right now the **hss_cx.xml** config file is applied to the current .rxf.

Download

To download the file, send a GET command (JSON format).

For example:

```
GET
http://localhost:8080/api/v1/sessions/0/ixload/test/activeTest/communityList/0/network/stack/childrenList/2/childrenList/3/childrenList/4/DiameterPortgroupData/download
```

The command can vary based on the test configuration.

The **200 OK** message received as the response will contain the Diameter .xml configuration file.

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 trafic, 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

You can use the IxLoad REST API to obtain the statistics generated during a test.

- L2-3 statistics sources become available after the test enters the running state and continue to be available after the test ends, until a new test is started or a new configuration is loaded. You cannot configure L2-3 statistics sources.
- L4-7 statistics sources become available when a new configuration is loaded. You can configure L4-7 statistics sources.

The root resource for statistics is the following URL:

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

To retrieve the list of statistics sources, send a GET request to this URL, as shown 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:

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

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw JSON Response

Copy to clipboard Save as file


```
{
  -0: { ... }
  -2000: { ... }
  -4000: { ... }
  -6000: {
      SSL Alerts Sent (unsupported_certificate): 0
      HTTP Client HTTP2.0 Continuation Frames Received: 0
      HTTP Client HTTP2.0 Connection Upgrade Succeeded: 0
      SSL Alerts Sent (certificate_expired): 0
      SSL Alerts Received (decryption_failed): 0
      HTTP Cookies Rejected - (Path Match Failed): 0
      Server Hello Received: 0
      SSL Alerts Received (bad_record_mac): 0
      Deflate Content-Encoding Decode Failed - Decoding Error: 0
      Client Hello Received: 0
```


Statistics views

REST API tests can display most of the statistics views that are currently displayed when running the IxLoad client in the GUI. The list of statistics views displayed in a test depends on the protocols enabled in the configuration.

Statistics views are displayed for both L2-3 and L4-7 protocols.

The REST Stat Views are available under the `stats` url:

`http://127.0.0.1:8080/api/v0/sessions/sessionId/ixload/stats/restStatViews`

Exceptions

The following types of statistics do not follow the same data format in REST as they do when viewed in the GUI. The views for these statistics types are either not shown in REST, or they are shown in aggregated form (not drilled-down):

- Per stream (for example, Video Client Per Stream)
- Per channel (for example, RTP Per Channel (VoIPSip))
- Per URL (for example, HTTP Client Per URL)

As in the GUI, the list of statistics views for the currently loaded configuration are only populated after the test enters the running stage. At that point, performing a GET on the `/restStatViews` URL will return the following result:

```
[
    {
        "caption": "HTTP Server - Transaction Rates",
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/0/statList",
                "rel": "statList"
            },
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/0/docs",
                "rel": "docs"
            },
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/0/values",
                "rel": "values"
            }
        ],
        "objectID": 0,
        "restObjectType": "ixStatView"
    },
    {
        "caption": "HTTP Server - Transactions",
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/1/statList",
                "rel": "statList"
            },
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/1/docs",
                "rel": "docs"
            },
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/1/values",
                "rel": "values"
            }
        ],
        "objectID": 1,
        "restObjectType": "ixStatView"
    },
```


Each statistics view object contains the list of statistics which are part of the view and the values for those statistics (values which are populated when the test is running). To view the list of statistics, navigate to the following link:

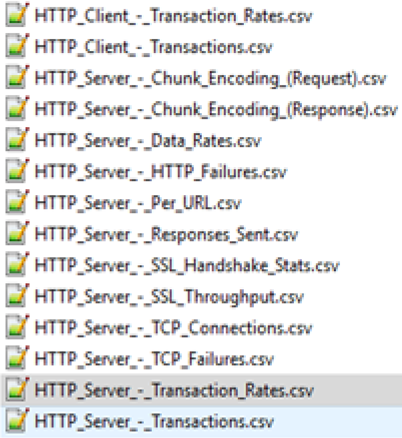
`http://127.0.0.1:8080/api/v0/sessions/sessionId/ixload/stats/restStatViews/statViewId/statList`


```
[
    {
        "objectID": 0,
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/0/statList/0/docs",
                "rel": "docs"
            }
        ],
        "caption": "Requests Received/s",
        "restObjectType": "ixStatViewStat",
        "aggregationType": "kRate",
        "statName": "HTTP Requests Received"
    },
    {
        "objectID": 1,
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload/stats/restStatViews/0/statList/1/docs",
                "rel": "docs"
            }
        ],
        "caption": "Requests Successful/s",
        "restObjectType": "ixStatViewStat",
        "aggregationType": "kRate",
        "statName": "HTTP Requests Successful"
    },
```

Enable stat view CSV logging

The stat values retrieved from `restStatViews` can be saved in csv format, in the results directory. This functionality is enabled by setting the `enableRestStatViewsCsvLogging` property on the `preferences` URL:

`http://127.0.0.1:8080/api/v0/sessions/sessionId/ixload/preferences`



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.

Video client per-stream statistics

For the IPTV Video Client activity, you can query the `VideoClientPerStream` stats from the REST API.

```
{
    "restObjectType": "ixRestStatController",
    "links": [
        {
            "href": "/api/v0/sessions/0/ixload/stats/IxServer",
            "rel": "IxServer"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoClientPerStream",
            "rel": "VideoClientPerStream"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/docs",
            "rel": "docs"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/RunState",
            "rel": "RunState"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoServer",
            "rel": "VideoServer"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoClientIPTVPerStream",
            "rel": "VideoClientIPTVPerStream"
        },
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoClient",
            "rel": "VideoClient"
        }
    ]
}
```

The values of the per-stream statistics can be retrieved by accessing the `@api/v0/sessions/0/ixload/stats/VideoClientPerStream/values` URL during the test run.

Based on their aggregation type, there are two types of per-stream statistics:

- `kString` – there is one value for each configured user (for example, the `Active` statistic):

```
{
    "objectID": 0,
    "links": [
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoClientPerStream/availableStats/0/docs",
            "rel": "docs"
        }
    ],
    "caption": "Active",
    "restObjectType": "ixAvailableStat",
    "aggregationType": "kString",
    "statName": "Active"
},
```

```
        "Data Rcvd": 10,
        "TVQM Int Avg Rel MOSV": 0,
        "Active": [
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
            "NO",
```

- `kSum` – there is one value for all users (for example, the `Stream Bit Rate` statistic):

```
{
    "objectID": 5,
    "links": [
        {
            "href": "/api/v0/sessions/0/ixload/stats/VideoClientPerStream/availableStats/5/docs",
            "rel": "docs"
        }
    ],
    "caption": "Stream Bit Rate",
    "restObjectType": "ixAvailableStat",
    "aggregationType": "kSum",
    "statName": "Stream Bit Rate"
},
```

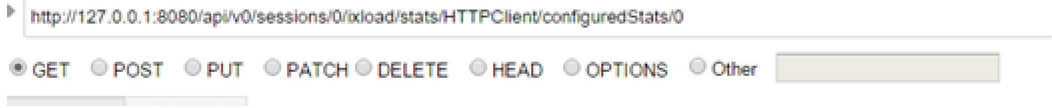
```
        "JB Packets Accepted": 0,
        "TVQM P Frames Impaired": 0,
        "Stream Bit Rate": 292500000,
        "I Frames Impaired": 0,
        "Gap Video Service Quality": 0,
        "Packets": 265033,
```

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:



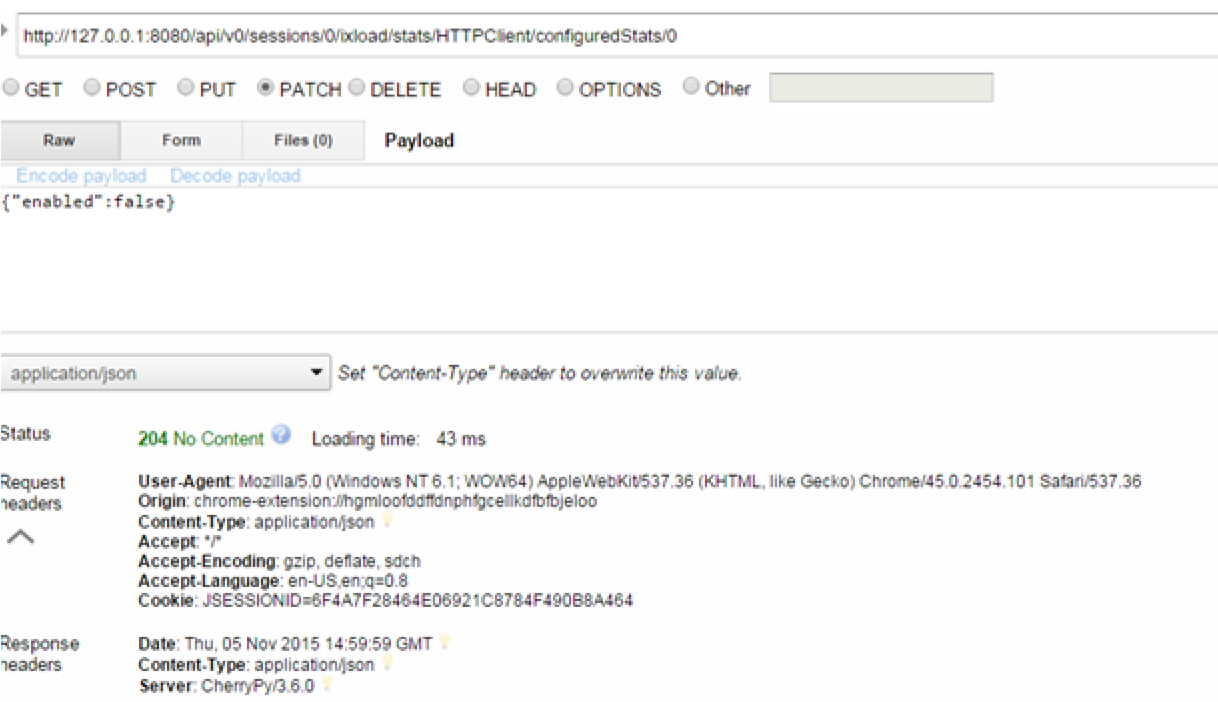
The result of get in the preceding request is shown in the following figure:

Raw | JSON | **Response**

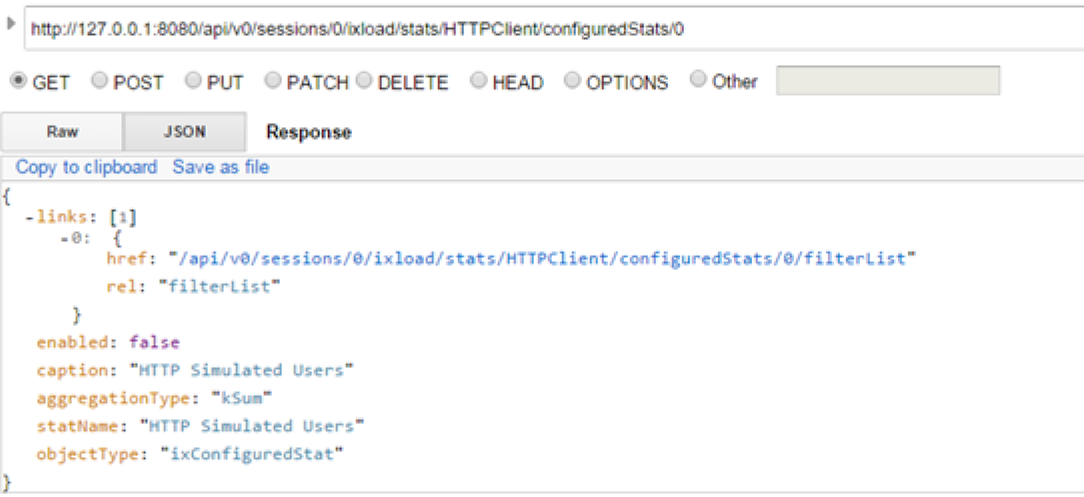
Copy to clipboard Save as file


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


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.

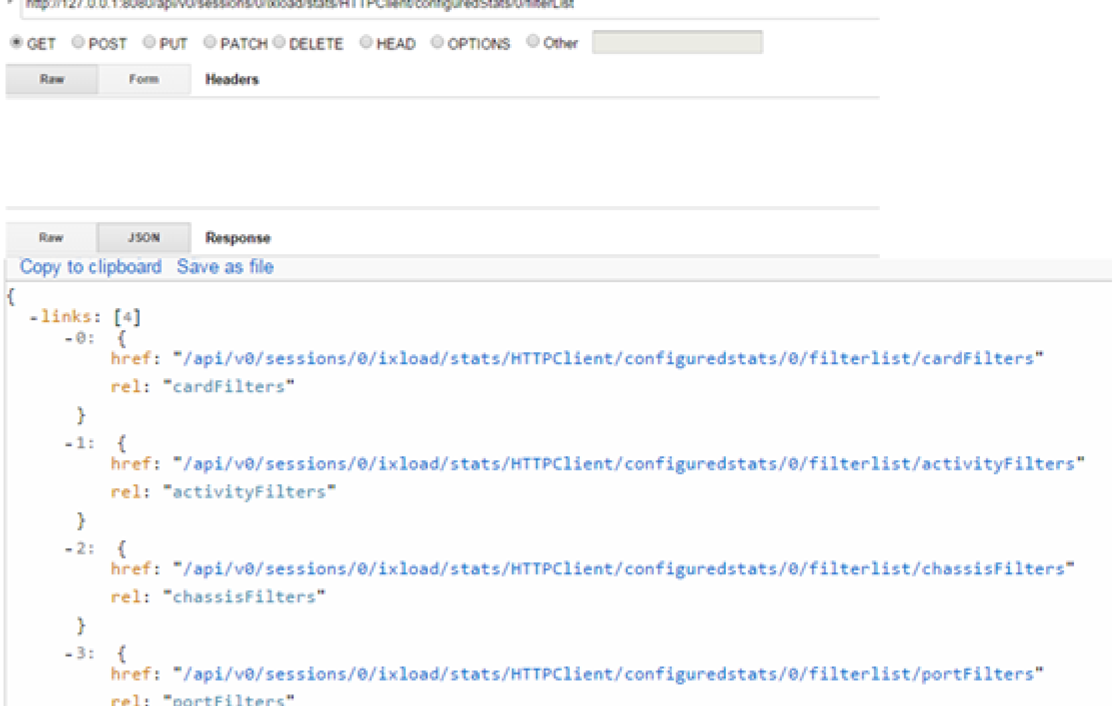


The following figure shows how the preceding PATCH method changed the configured statistics structure by turning it off:



Filtering stats

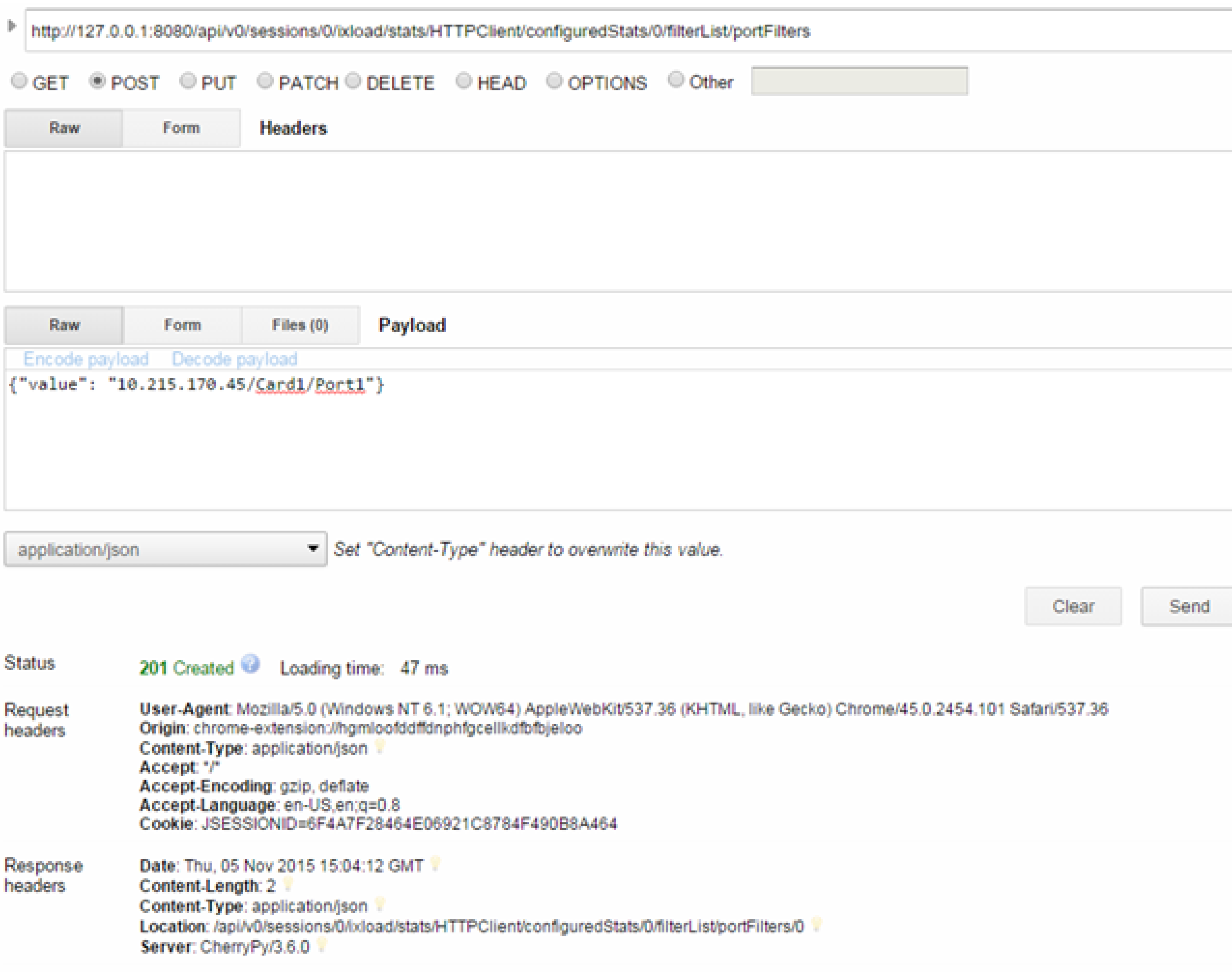
The filter statistics are accessed as shown in the following figure by issuing a GET command on the filter list from a specific configuredStat item:



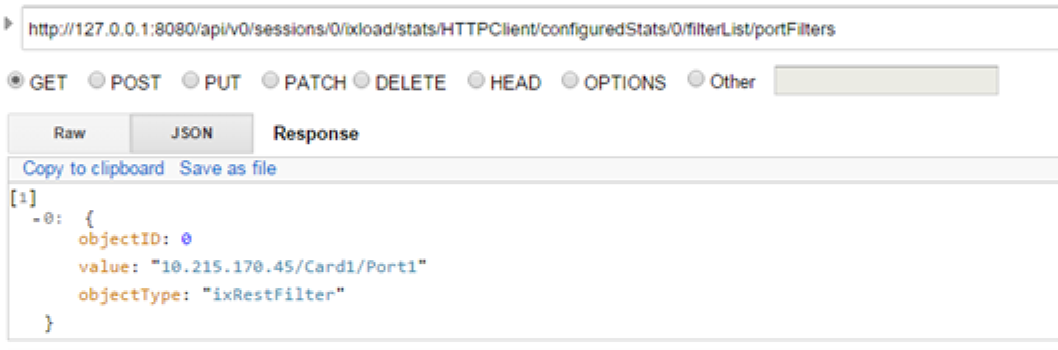
A configured statistic contains several filters, each enabling the option to get the values at:

- 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 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"

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"
      }
    ],
    "restObjectType": "ixRestLogEntry",
    "message": "Validating that 10.215.122.22 accepts incoming connections. Will try to connect for 10 seconds."
  },
  {
    "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": "Validation for 10.215.122.22 is completed. IP is valid and connection succeeded."
  },
```

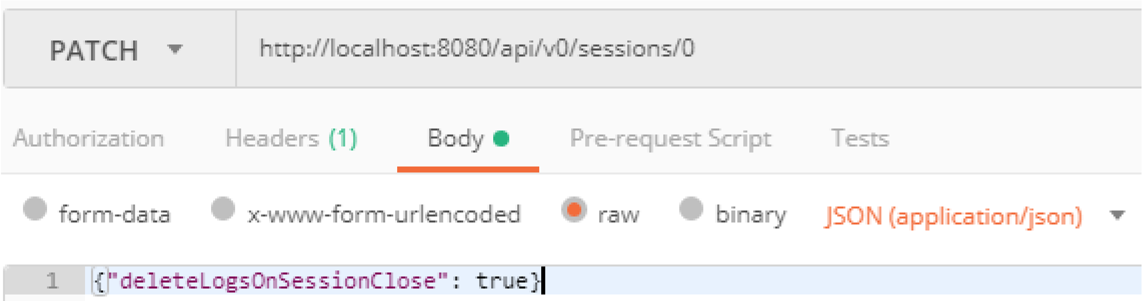

Deleting logs

Each session object has a property names `deleteLogsOnSessionClose` that is set to `false` by default.

```
[
    {
        "ixLoadVersion": "8.50.0.165",
        "activeTime": "0:20:51",
        "backendType": "ixload",
        "links": [
            {
                "href": "/api/v0/sessions/0/ixload",
                "rel": "ixload"
            },
            {
                "href": "/api/v0/sessions/0/docs",
                "rel": "docs"
            }
        ],
        "objectID": 0,
        "deleteLogsOnSessionClose": false,
        "remotePid": 11224,
        "sessionId": 0,
        "applicationType": "ixload",
        "restObjectType": "ixSession",
        "id": 0,
        "isActive": true
    }
]
```

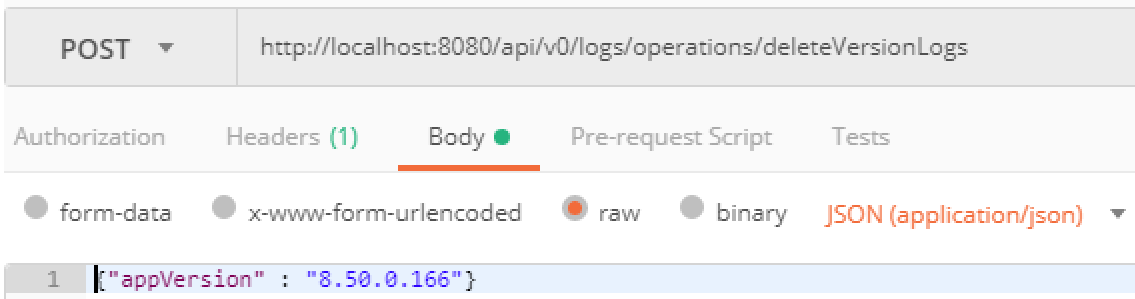

Deleting logs for an instance

By performing a PATCH operation on the session URL, with the `{"deleteLogsOnSessionClose" : true}` payload, all session logs (the `IxLoadRest-x-yy.log` and all client logs for the IxLoad used by the session) will be deleted when the session is deleted (using the DELETE operation).



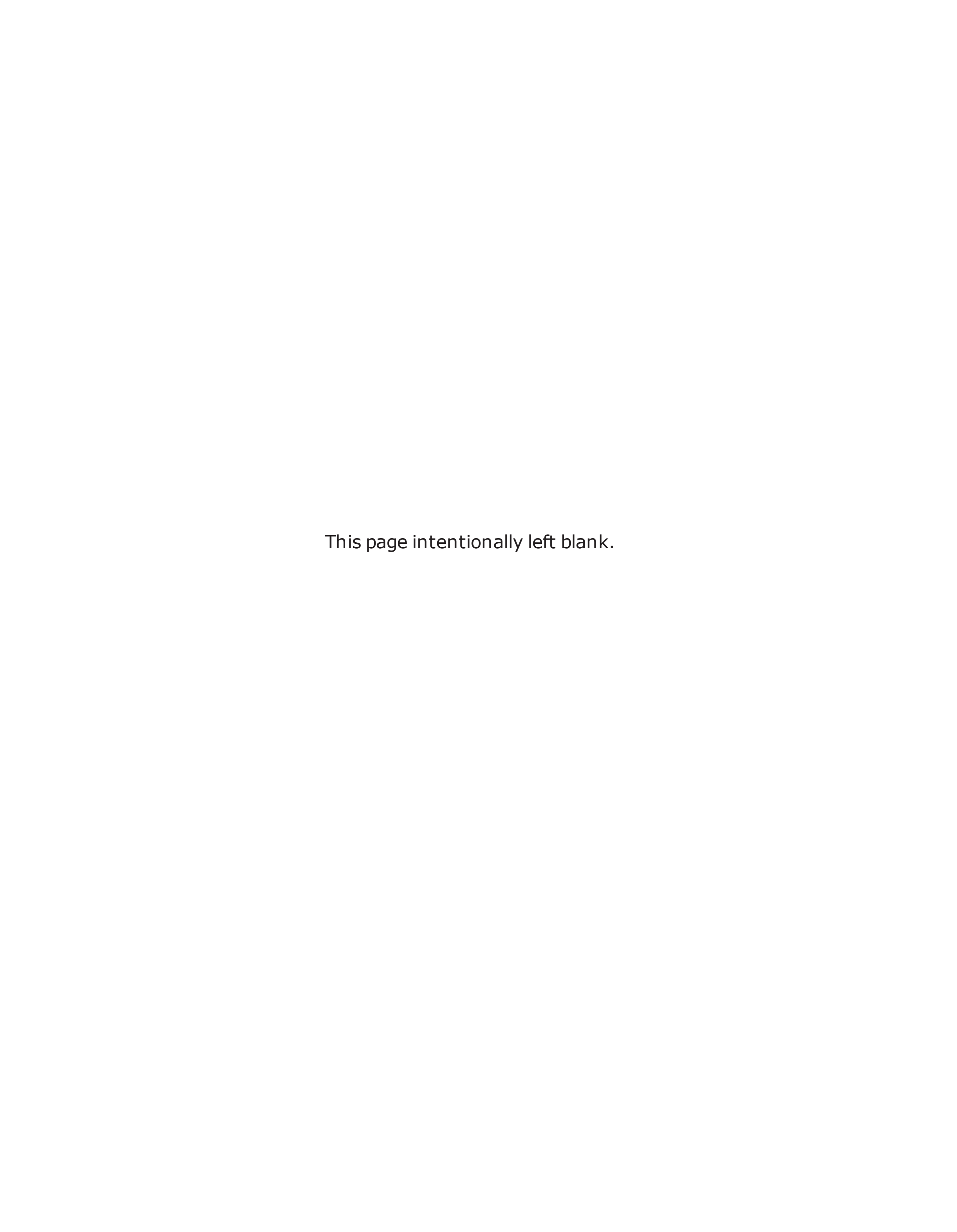
Deleting logs for a specific IxLoad version

By performing a POST operation on the `@api/v0/logs/operations/deleteVersionLogs` URL with the payload `{"appVersion" : IxLoad-version}` and header `{"content-type": "application/json"}` , all the logs for the specified version of IxLoad will be deleted (all `IxLoadRest-x-yy.log` files and all client logs resulted from all the sessions that used the specified version of IxLoad).



Deleting logs for all IxLoad versions

By performing a POST operation on the `@api/v0/logs/operations/deleteAllLogs` URL without any payload, but with the header `{“content-type”: ”application/json”}`, all the logs for all IxLoad versions present on the machine will be deleted (all `IxLoadRest-x-yy.log` files and all client logs).



This page intentionally left blank.

REST Script Templates

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.

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 a number of Python scripts.The scripts must 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`.

You can use Python 2.7 or Python 3 to run the scripts.

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.py

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.py

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.py

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.

CIFSfromScratch.py

This template creates a CIFS scenario starting from an empty configuration and runs it.

Dhcpv4v6_config_from_scratch.py

This template creates a DHCP configuration and runs it.

DNS_with_DUT_from_scratch.py / DNS_config_from_scratch.py

This template creates a DNS scenario starting from an empty configuration and runs it.

FTP_config_from_scratch.py

This template creates an FTP scenario starting from and empty configuration and runs it.

HTTP_ssl_ipsec_ipv4v6_config_from_scratch.py

This template creates an HTTP over IPSEC configuration and runs it.

IMAP_config_from_scratch.py

This template creates an IMAP scenario starting from an empty configuration and runs it.

POP3ConfigFromScratch.py

This template creates a POP3 scenario starting from an empty configuration and runs it.

RepRunner.py

This template runs a set of repositories in the same IxLoad session, one after the other.

RTSP_config_from_scratch.py

This template creates a RTSP scenario starting from an empty configuration and runs it.

SimpleRun.py

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.

SimpleRunCapturesEnabled.py

This template enables Analyzer on ports before starting a test.

After the test stops and the capture files are received from the ports, it downloads the captures locally.

SMTPfromScratch

This template creates a SMTP scenario starting from an empty configuration and runs it.

StatelessPeerFS.py

This template creates a Stateless Peer scenario starting from an empty configuration and runs it.

TFTP_config_from_scratch.py

This template creates a TFTP scenario starting from an empty configuration and runs it.

VoIPSIP_config_from_scratch.py

This template creates a VoIP SIP scenario starting from an empty configuration and runs it.

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.

addChassisList

Adds one or more chassis to the chassis list.

Syntax: `addChassisList(connection, sessionUrl, chassisList)`

Arguments:

`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

Adds commands to a certain list of provided agents.

Syntax: `addCommands(connection, sessionUrl, commandDict`

Arguments:

`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 }] })

addDUT

Adds a DUT resource to the active test on the given session.

Returns the ID of the newly added DUT.

Syntax: `addDUT(connection, sessionUrl, dutDict=None)`

Arguments:

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

`dutListUrl` is the address that contains the list of DUTs

`dutDict` contains a comment, the name or the type of the DUT (or all three)

DUT types:

Firewall

ExternalServer

PacketSwitch

ServerLoadBalancer

VirtualDut

By default, when posting using dutDict=None, dutType will be SLB

assignPorts

Assigns ports from a connected chassis to the required NetTraffics.

Syntax: `assignPorts(connection, sessionUrl, portListPerCommunity)`

Arguments:

`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

Changes certain properties for the provided activities.

Syntax: `changeActivityOptions(connection, sessionUrl, activityOptionsToChange)`

Arguments:

`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

Changes the interface mode on a list of cards from a chassis. To call this method, the required chassis must be already added and connected.

Syntax: `changeCardsInterfaceMode (connection, chassisChainUrl, chassisIp, cardIdList, mode)`

Arguments:

`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

Changes certain properties on an IP Range.

Syntax: `changeIpRangesParams(connection, sessionUrl, ipOptionsToChangeDict)`

Arguments:

`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

Clears all commands from the command list of the agent names provided.

Syntax: `clearAgentsCommandList(connection, sessionUrl, agentNameList)`

Arguments:

`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

Clears the chassis list. After execution, no chassis should be available in the chassisList.

Syntax: `clearChassisList(connection, sessionUrl)`

Arguments:

`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.

collectDiagnostics

Performs a POST request to collect log files and packages them into a ZIP file.

Syntax: `collectDiagnostics(connection, sessionUrl, zipFilePath, clientOnly=False)`

Arguments:

`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 collect diagnostics for

`zipFilePath` is the local zip path on the machine that holds the IxLoad instance

collectGatewayDiagnostics

Performs a POST request to collect gateway log files and packages them into a ZIP file.

Syntax: `collectGatewayDiagnostics(connection, zipFilePath)`

Arguments:

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

`zipFilePath` is the local zip path on the machine that holds the IxLoad instance

createSession

Creates a new session. The return value is the URL of the new session.

Syntax: `createSession(connection, ixLoadVersion)`

Arguments:

`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

Deletes an existing session.

Syntax: `deleteSession(connection, sessionUrl)`

Arguments:

`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.

editDutConfig

Modifies the settings found in the `dutConfig` page and its subpages.

The return value is a dictionary with the reply from the server for patch/delete and the `objectId` for post actions as a value, and the corresponding `networkDict` as a key.

Syntax: `editDutConfig(connection, dutUrl, configDict)`

Arguments:

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

`dutUrl` is the address of the dut that needs to be changed/modified

`configDict` is a list that contains the actions needed to be performed on the target DUT, and dictionaries with the information required for every action

Example dictionary:

```
{
            "post":
            {
                "originateNetwork.<arbitraryIdentifier1>": {}
                "originateNetwork.<arbitraryIdentifier2>":
                {
                    "ipCount": "200",
                    "firstIp": "10.10.10.10"
                }
            }
            "patch":
            {
                "terminateNetwork.<validObjectId1>":
                {
                    "ipCount": "500"
                }
```

```
        }
    }
```

Format for network/protocol names:

Server Load Balancer: `slb.<identifier>`

Packet Switch: `originateNetwork.<id>, terminateNetwork.<id>, terminateProtocolPort.<id>, originateProtocolPort.<id>`

Virtual DUT: `network.<id>, protocolPort.<id>`

editDutProperties

Modifies the DUT's name, comment, and type.

Syntax: `editDutProperties(connection, sessionUrl, dutId, newInfoDict=None)`

Arguments:

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

`dutUrl` is the address of the dut that needs to be changed/modified

`newInfoDict` is a dictionary that contains the updated DUT information

enableAnalyzerOnPorts

Enables Analyzer for a specific port on a specific community.

Syntax: `enableAnalyzerOnPorts(connection, sessionUrl, communityPortIdTuple)`

Arguments:

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

`communityPortIdTuple` is a tuple composed of (communityID and portName)

`communityID` is the id of the community list for which captures should be retrieved.

`portName` is the name of the port for which Analyzer will be enabled (in the format 'n.n.n', not 'Port n.n.n')

`sessionUrl` is the address of the session on which the test was run.

getCommandListUrlForAgentName

Gets the `commandList` url for a provided agent name.

Syntax: `getCommandListUrlForAgentName(connection, sessionUrl, agentName)`

Arguments:

`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

Returns the IP Ranges associated with an IxLoad network component.

Syntax: `getIPRangeListUrlForNetworkObj(connection, networkUrl)`

Arguments:

`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

Gets the test current state (for example: running, unconfigured).

Syntax: `getTestCurrentState(connection, sessionUrl)`

Arguments:

`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

Gets the error that appeared during the last test run.

If no error appears, the test ran successfully and the return value is `None`.

Syntax: `getTestRunError(connection, sessionUrl)`

Arguments:

`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

Performs a POST request to load a repository.

Syntax: `loadRepository(connection, sessionUrl, rxfFilePath)`

Arguments:

`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

Performs a generic DELETE method on a given URL.

Syntax: `performGenericDelete(connection, listUrl, payloadDict)`

Arguments:

`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

Performs a generic operation on the given URL, and waits for it to finish.

Syntax: `performGenericOperation(connection, url, payloadDict)`

Arguments:

`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

Performs a generic PATCH method on a given URL.

Syntax: `performGenericPatch(connection, url, payloadDict)`

Arguments:

`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

Performs a generic POST method on a given URL.

Syntax: `performGenericPost(connection, listUrl, payloadDict)`

Arguments:

`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

Polls for statistics. Polling statistics is per request, but this method does a continuous poll.

Syntax: `pollStats(connection, sessionUrl, watchedStatsDict, pollingInterval=4)`

Arguments:

`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` are the statistics that are being monitored.

`pollingInterval` is the polling interval. The default is 4 but can be overridden.

retrieveCaptureFileForPorts

Retrieves capture files from a REST session that had `portCapture` set to `True`.

Syntax: `retrieveCaptureFileForPorts(connection, sessionUrl, communityPortIdTuple, captureFile)`

Arguments:

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

`communityPortIdTuple` is a tuple composed of (`communityID` and `portName`)

`communityID` is the ID of the community list for which captures should be retrieved.

`portName` is the name of the port for which capture will be enabled (in the format 'n.n.n', not 'Port n.n.n')

`sessionUrl` is the address of the session on which the test was ran.

`captureFile` is the save path for the capture file

Error Codes:

0 No error

1 Invalid `portId`

2 Cannot create/open `captureFile`

runTest

Starts the currently loaded test. After starting the 'Start Test' action, wait for the action to complete.

Syntax: `runTest(connection, sessionUrl)`

Arguments:

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

Saves the current rxf to the disk of the computer on which the IxLoad instance is running.

Syntax: `saveRxf(connection, sessionUrl, rxfFilePath)`

Arguments:

`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

Changes the aggregation mode on a list of cards from a chassis. To call this method, the required chassis must be already added and connected.

Syntax: `setCardsAggregationMode(connection, chassisChainUrl, chassisIp, cardIdList, mode)`

Arguments:

`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.

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.

Arguments:

`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.’

waitForActionToFinish

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 method 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.

Syntax: `waitForActionToFinish(connection, replyObj, actionUrl)`

Arguments:

`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` is the URL pointing to the operation.

waitForAllCaptureData

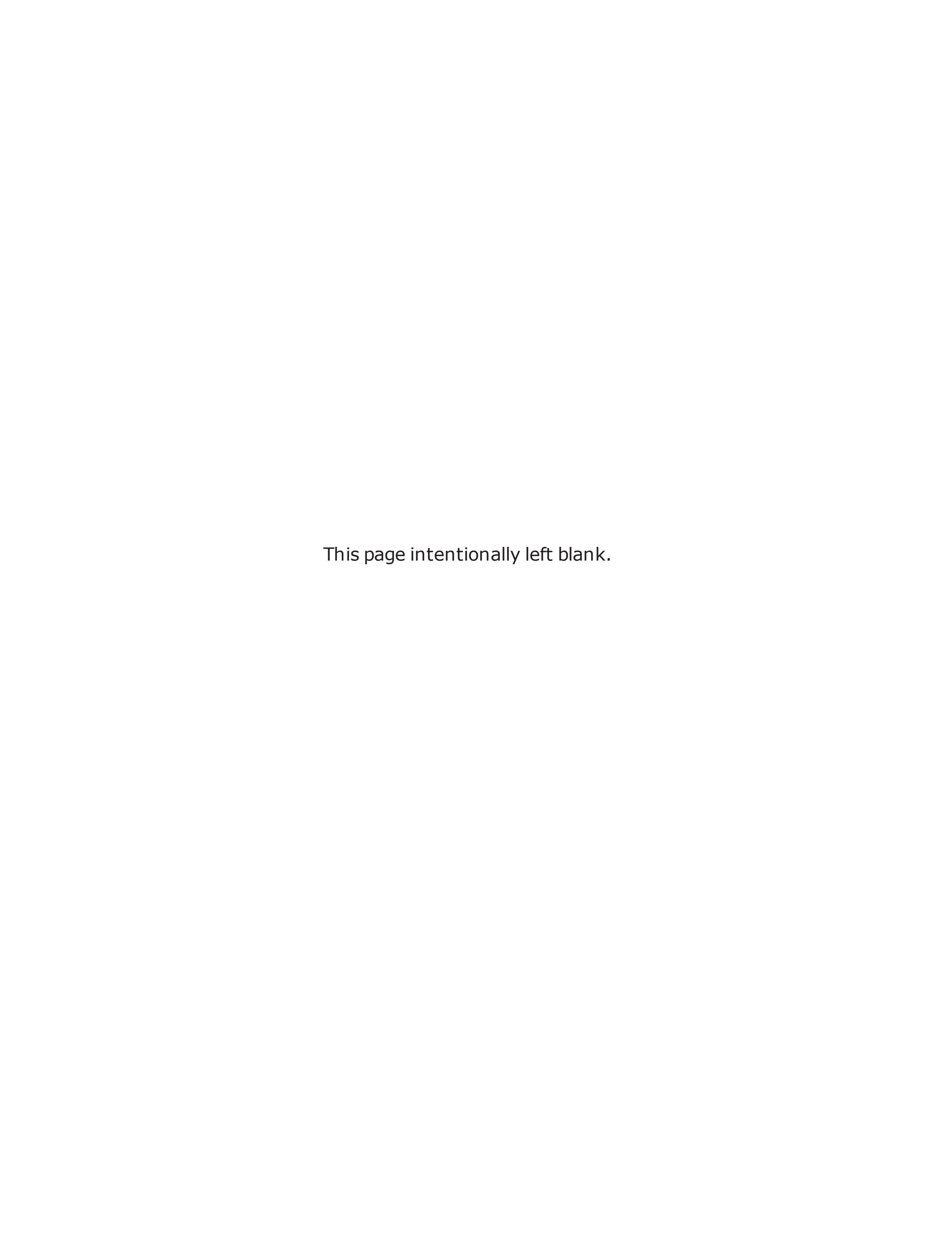
This method is used to wait for the test to capture all the port data that was received after the test has finished running.

Syntax: `waitForAllCaptureData(connection, sessionUrl)`

Arguments:

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.



This page intentionally left blank.

Ixia
26601 West Agoura Road
Calabasas, California 91302