



IxLoad

REST API Programming Guide

Version 9.00 Update 1

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
<i>Regional and local support contacts:</i>		
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	xii
Before you Begin	xiv
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	24
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
findURLs operation	36
IxLoad Session Handling	41
Creating a new session	41
New session with a specified version	41
New session with the latest version	45
Deleting a session	46
Uploading and downloading files	47
API Browser	49
How to find URLs in a REST API session	52
IxLoad Data Model	55
Communities	55
Timelines	56
Login name	56
DUTs	56
Expiration timer	58
Enabling Analyzer and downloading captures	59

Modifying the activity user objective value on the fly	60
Chassis Chain/Port Assignment Operations	61
Adding a chassis	61
Connecting to a chassis	62
Removing a chassis	64
Assigning ports	64
Taking or clearing ownership of ports	66
Rebooting ports	66
Unassigning ports	66
IxVM chassis (ixChassisBuilder)	66
Upload and Download Diameter XML Configuration Files	71
Statistics	73
Viewing statistics	73
Statistics views	76
RunState stat source	78
Video client per-stream statistics	78
Modifying configured statistics	80
Filtering stats	82
Generated CSVs	84
Logging	85
REST Script Templates	89
AddNewCommand.py	89
ChangeAgentObjectives.py	90
ChangeIpType.py	90
CIFSfromScratch.py	91
Dhcpv4v6_config_from_scratch.py	91
DNS_with_DUT_from_scratch.py / DNS_config_from_scratch.py	91

FTP_config_from_scratch.py	91
HTTP_ssl_ipsec_ipv4v6_config_from_scratch.py	91
IMAP_config_from_scratch.py	91
POP3ConfigFromScratch.py	91
RepRunner.py	91
RTSP_config_from_scratch.py	91
SimpleRun.py	91
SimpleRunCapturesEnabled.py	92
SMTPfromScratch	92
StatelessPeerFS.py	92
TFTP_config_from_scratch.py	92
VoIPSIP_config_from_scratch.py	92
IxLoadRestUtils	93
class Connection(__builtin__.object)	93
class WebList(__builtin__.list)	94
class WebObject(__builtin__.object)	95
Functions	95
IxLoadUtils	97
addChassisList	97
addCommands	97
addDUT	97
assignPorts	98
changeActivityOptions	98
changeCardsInterfaceMode	98
changeIpRangesParams	99
clearAgentsCommandList	99
clearChassisList	99

collectDiagnostics	100
collectGatewayDiagnostics	100
createSession	100
deleteSession	100
editDutConfig	101
editDutProperties	102
enableAnalyzerOnPorts	102
getCommandListUrlForAgentName	102
getIPRangeListUrlForNetworkObj	102
getTestCurrentState	103
getTestRunError	103
loadRepository	103
performGenericDelete	103
performGenericOperation	104
performGenericPatch	104
performGenericPost	104
pollStats	105
retrieveCaptureFileForPorts	105
runTest	105
saveRxf	106
setCardsAggregationMode	106
uploadFile	106
waitForActionToFinish	107
waitForAllCaptureData	107

This page intentionally left blank.

New in this Release

The following features are new in this release:

findURLs	A new operation, <code>findURLs</code> , enables you to find resources in a test configuratin. See findURLs operation on page 36 .
Starting a new session	You can start a new session from the API Browser. See Creating a new session on page 41 .

This page intentionally left blank.

Before you Begin

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

Authentication

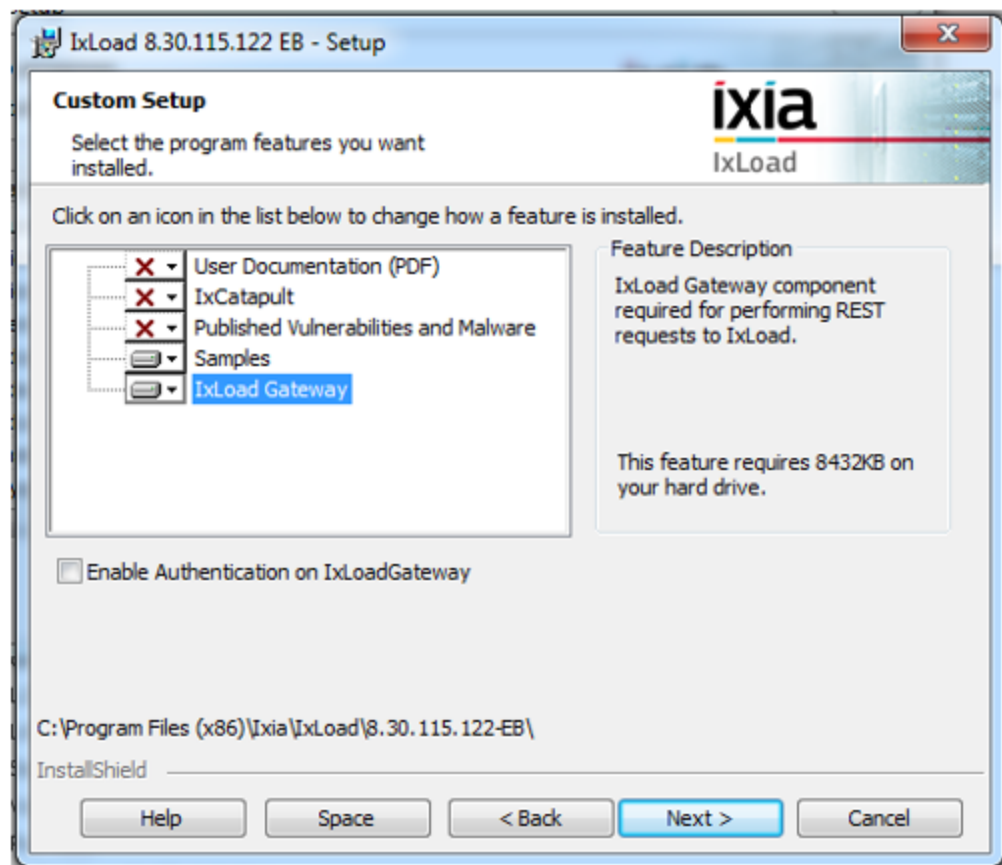
If you want to use the REST API with authentication, an Ixia User Management server must be available on the network. See [REST Authentication on page 9](#) for more information.

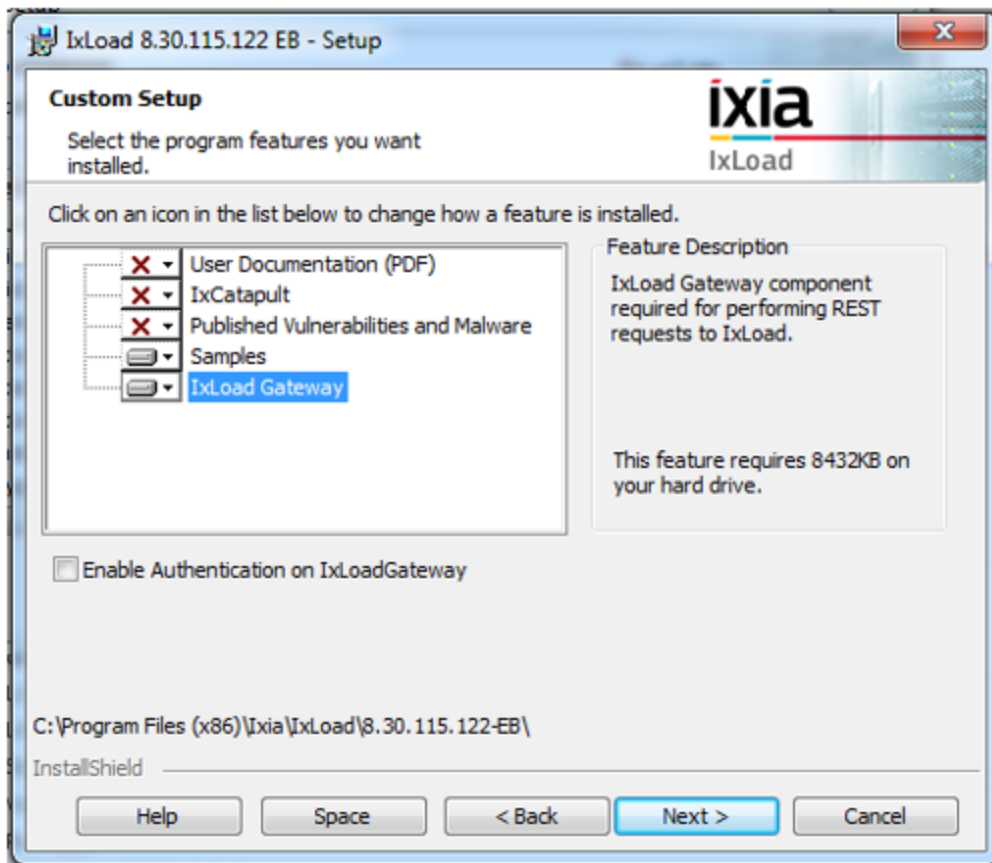
Gateway Service

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, you can modify the installation to add it: 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.





This page intentionally left blank.

REST Resources

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

A resource is a basic concept of a REST API. In the IxLoad REST API, a resource is a representation of an IxLoad object for the user. Not all IxLoad objects are resources, and not all IxLoad object functionality is available through the REST API.

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

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) from a local path.
- View the data model tree through GET requests (including query string support).
- Add a new chassis or remove an existing chassis.
- Assign and unassign ports.
- Change the existing configuration and modify field values through PATCH requests.
- Support for L2-3 ranges.
- Support for L4-7 plugins.
- Save a configuration modified through the REST API.
- View, add, or delete the configured L2-3 and L4-7 statistics.
- Run a 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) or removing (DELETE) objects such as test communities, plugins, and ranges. Add and remove operations are only supported on the chassis list, the port lists, and the configured statistics.
- Creating or editing voice scenarios
- Creating or editing Diameter scenarios (except for importing or 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	v1
<pre>"links": [{ "href": "/api/v0/sessions/1/ixload" "rel": "ixload" }, { "href": "/api/v0/sessions/1/docs", "rel": "docs" }], "MethodType": "ixload"</pre>	<pre>"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" }], "MethodType": "ixload"</pre>

applicationTypes response

A GET on api/vX/applicationtypes will return:

v0	v1
<pre>{ "appName": "8.40.0.277" }</pre>	<pre>{ "name": "8.40.0.277" }</pre>

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
<pre>{"ixLoadVersion":"8.50.0.75"}</pre>	<pre>{"applicationVersion":"8.50.0.75"}</pre>

204 No Content response

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

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

Using the REST API over HTTPS

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`: The actual self-signed certificate.
- `ixload_privkey.key`: 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: `<IxLoadGateway_Install_Path>\certificate`
- On the IxLoad Linux OVA: `/opt/ixia/ixloadgateway/certificate`

The self-signed certificate is generated by using a 2048-bit RSA key pair 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, the 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 script 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 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.

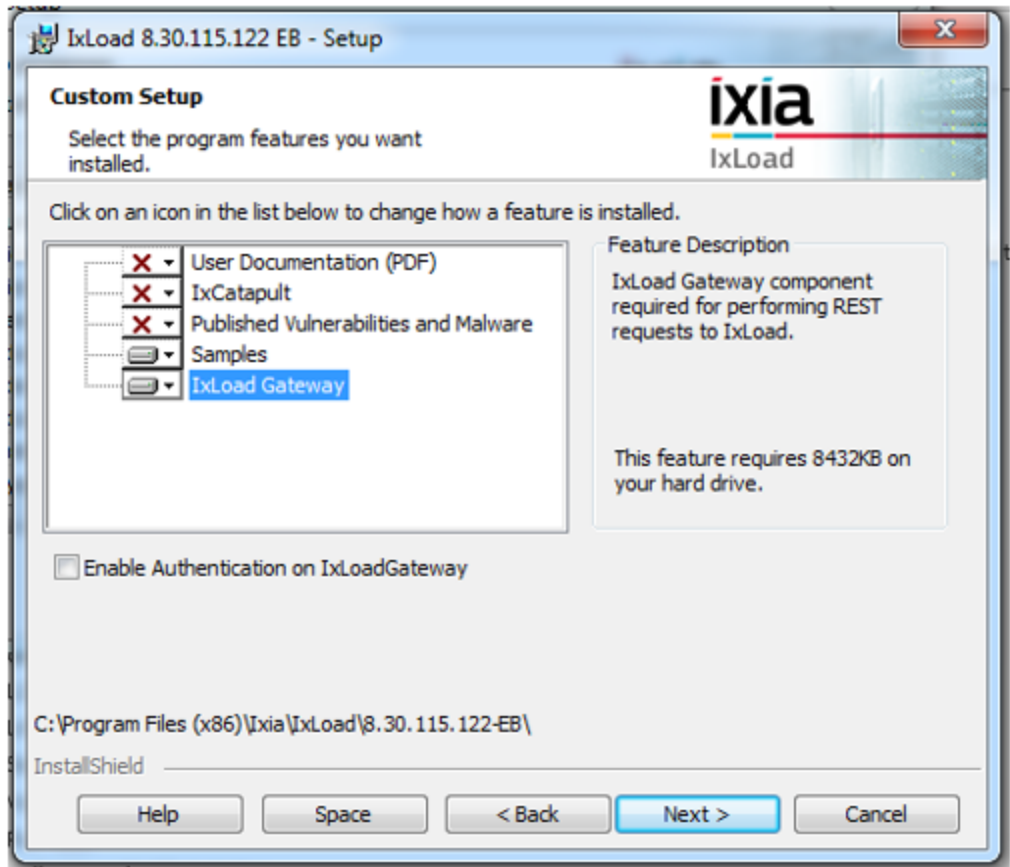
User Management is a standalone application that you can download from the IxLoad section of Ixia's website (<https://support.ixiacom.com/support-overview/product-support/downloads-updates/versions/33>).

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

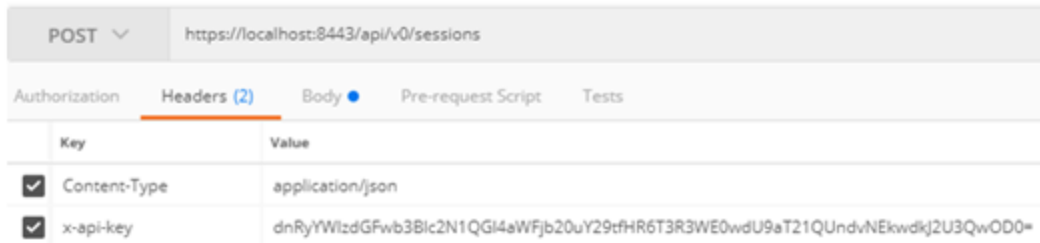
To enable or disable authentication on Linux, run the `configRestAuth.sh` script in `/opt/ixia/ixloadgateway` with the following switches:

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

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

Authenticating REST requests

Most request headers contain an `api-key`. An `api-key` is generated by the user management component based on a username and password pair. As a result, most requests need to have an `api-key` present in their headers. The following figure shows an example of an `api-key`:



Key	Value
<input checked="" type="checkbox"/> Content-Type	application/json
<input checked="" type="checkbox"/> x-api-key	dnRyYWiZdGFwb3Blc2N1QGI4aWFjb20uY29tHR6T3R3WE0wdU9aT21QUndvNEkwdk2U3QwOD0=

The only requests that do not need to contain an `api-key` are:

- 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 because it 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 because it is 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"  
}
```

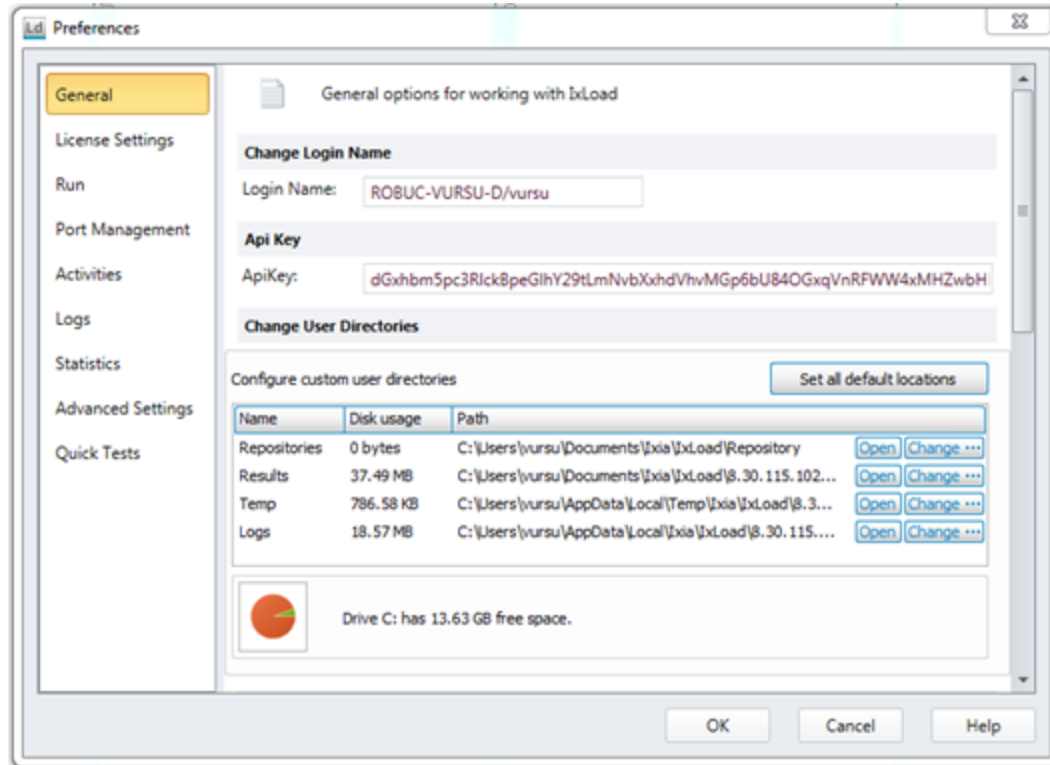
Users can delete only their own sessions (sessions that were created with the same `api-key` as the one provided during the DELETE request).

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 be made to IxLoad REST scripts for authentication are as follows:

- `kApiKey = ''`: If this value remains an empty string, the `api-key` will not be included in request headers. Otherwise, it will be included in the request headers.

- `connection.setApiKey(kApiKey)` : Sets the `api-key` for the connection.

This page intentionally left blank.

Supporting Methods and Running Operations

This section describes how resources are represented, how they are accessed and changed, and the exposed data model in the IxLoad REST API.

REST representation

The Ixload REST API handles many different object types. Each object has among its values the following:

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

Primitive values

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

- Strings are enclosed in quotes. For example: `"custom string," ""`
- Numbers, integers, or float are not enclosed in quotes. For example: `1, 1.1`
- Booleans are not enclosed in quotes, and are all lowercase. For example: `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 with it, 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. This returns the `objectID`.

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

For example, for a list with 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
```

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, except for the `api` string at the beginning of a 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 the 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": [{"link": "http://127.0.0.1:8080/api/v0/sessions/0/ixload/preferences"}],
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. Because all the global options except for Maximum Instances, License Model, and License Server are saved per-user, this means that settings made in the IxLoad UI have 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 supported for payloads is JSON. The payload applies to PATCH, POST, and DELETE methods.

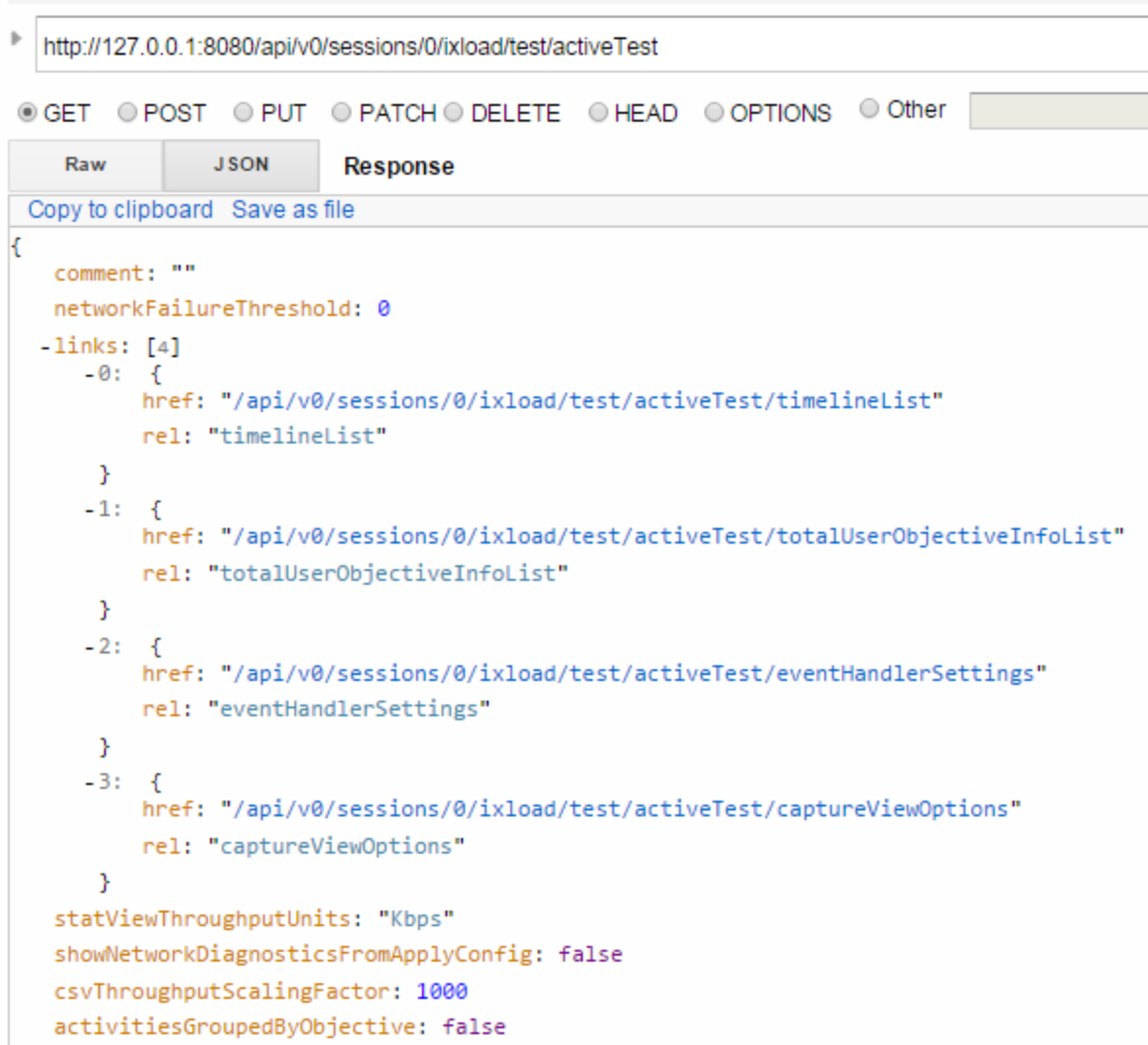
GET

A GET request receives the list of REST options for the requested resource. The GET request does not contain a 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 request applied to the `activeTest` REST resource in IxLoad:



The preceding figure shows the output of a GET made through the Advanced REST Client in Google Chrome. The actual representation will be different depending to the programming language used to access the IxLoad REST API.

PATCH

A PATCH request changes field values on resources exposed by the IxLoad session. The PATCH request receives as its payload a list of options that you can 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 to be changed. This means one `field name:new value` pair. The following figure shows a PATCH method made from 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

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

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

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

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

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

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

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

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

DELETE

A DELETE request deletes one or more of the elements of the list. If the request is successful, a **204 No Content** status is returned.

DELETE requests do not require a 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 to 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

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


OPTIONS requests do not require a payload.


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 the output of an OPTIONS request in the Advanced REST client:

▶

☐ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☒ OPTIONS ☐ Other



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

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

Operations

In addition to the HTTP requests described in [IxLoad REST methods on page 17](#) that are executed on basic resources (objects or lists of the IxLoad data model), the IxLoad REST API also offers support for operations. These are asynchronous actions performed on a certain resource (a URL) that change the resource's state. They do not add, remove, or change the field values of the resources that they are applied to. Some examples of operations are: starting an inactive IxLoad session, connecting to an existing chassis, or running a test.

To find the operations that are available for a certain resource, perform a GET request on the resource URL, with `/operations` added to the end of the URL. For example, the following figure shows the operations available for the `test` REST resource:



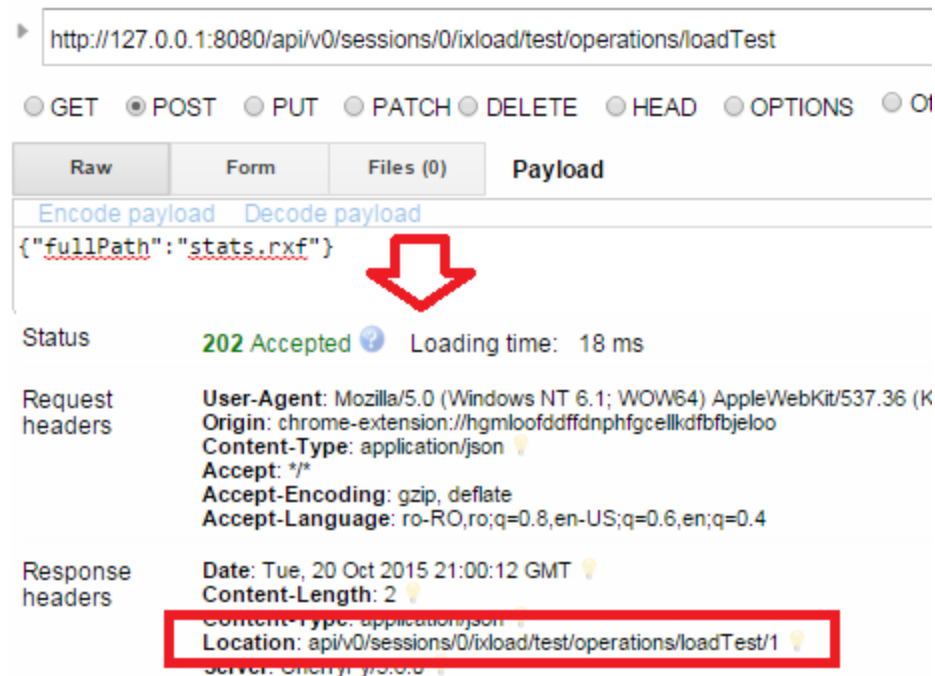
The GET response contains the operations that are available, the parameters that they require, and the default values for the parameters.

Starting an operation

To start an operation, perform a POST request on the following URL:

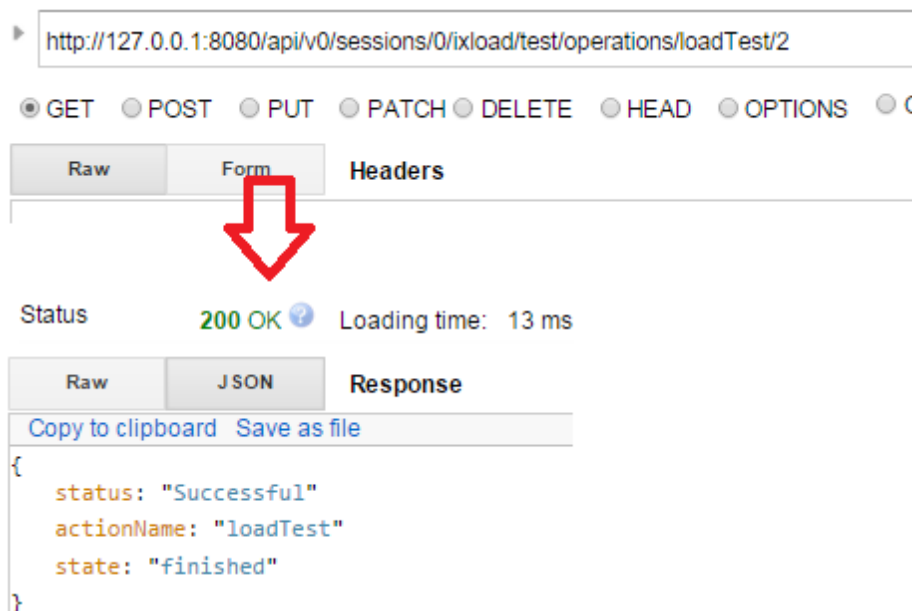
`$resourceUrl/operations/$operationName`

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 a 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. To do this, when you start an operation (that is, execute the POST request), the response header includes a field called **Location** that contains a URL. If you perform a GET request on that URL, the operation's status will be returned. The following figure shows the output for getting the operation status for the `loadTest` operation:



The following table lists the possible values for the `state` and `status` fields:

state	Created: the operation was created. Executing: the operation is in progress. Finished: the operation is complete.
status	Not started: the operation has not started yet. Operations are synchronous, and the operation might be waiting for other operations to finish. In Progress: the operation is being executed.

If the operation fails (exits with an error), a new field is included in the preceding response that contains the error message returned by the operation. For example:

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

Important! The URL that retrieves an operation's status has a lifetime of 10 minutes. If you perform a GET request on an operation `URL/operationID` URL after this lifetime has expired, the REST API returns a 400 Bad Request error.

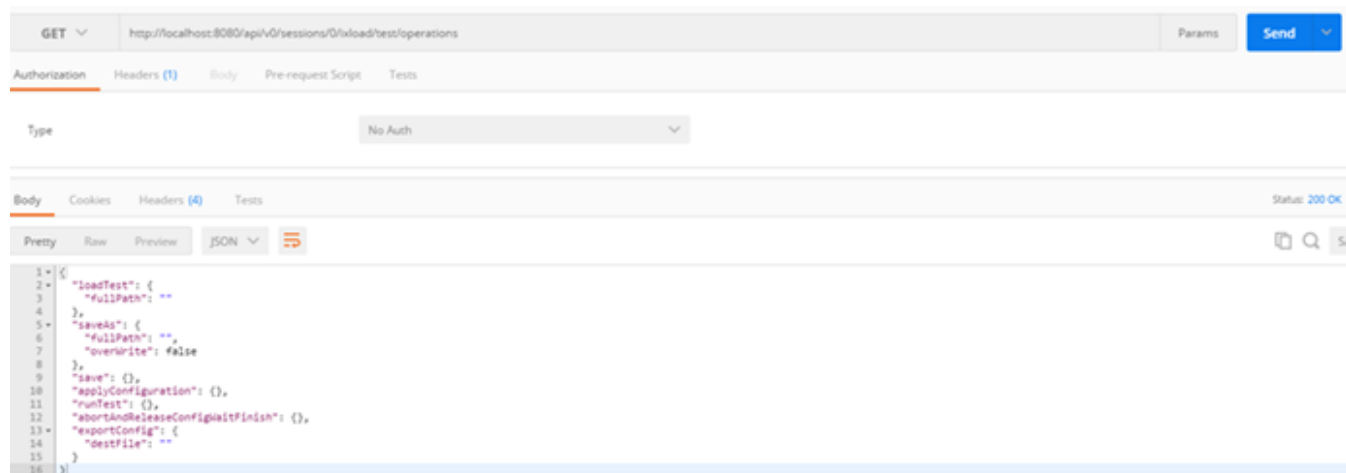
Examples of common operations in the IxLoad REST API

A list of the most commonly used operations for an IxLoad test in the REST API can be obtained by performing 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):

<code>abortAndReleaseConfigWaitFinish</code>	Stop the currently running IxLoad test.
<code>applyConfiguration</code>	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.
<code>exportConfig</code>	Export the currently loaded configuration file as a .crf file. The location of the archive needs to be passed as a parameter.
<code>importConfig</code>	Import a .crf file as the current test configuration. The location of the .crf file and the location where the .rxp file will be saved after the import must be passed as parameters.
<code>loadTest</code>	Load an IxLoad configuration file. The <code>fullPath</code> of the rxp file to be loaded must be passed as a parameter.
<code>runTest</code>	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.
<code>save</code>	Save the currently loaded configuration file.
<code>saveAs</code>	Save the currently-loaded configuration file as a new file. The new file path for the rxp must be passed as a parameter, and the <code>overwrite</code> option in case the file already exists.
<code>waitForAllCaptureData</code>	Wait for the test to capture all the port data that was received after the test has finished running.



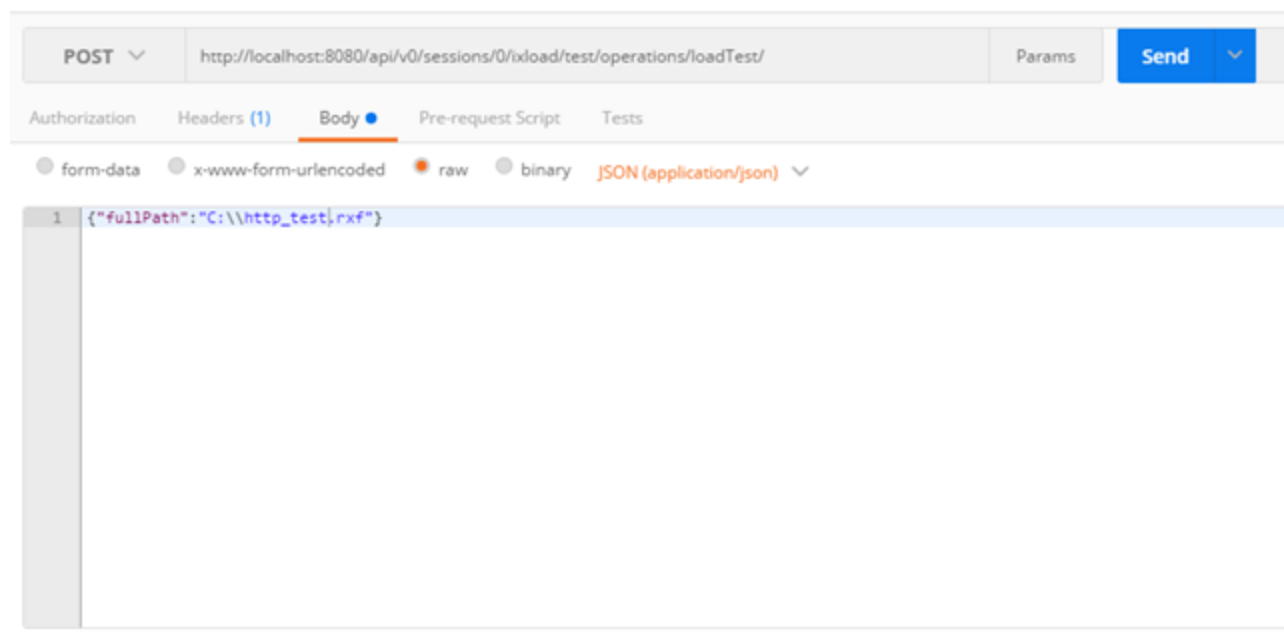
Example of loading a repository (.rxf) file

On an active session, do a POST on a URL similar to the following:

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

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

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



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

Example of importing a .crf file

On an active session, do a POST on an URL similar to the following:

Operations

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

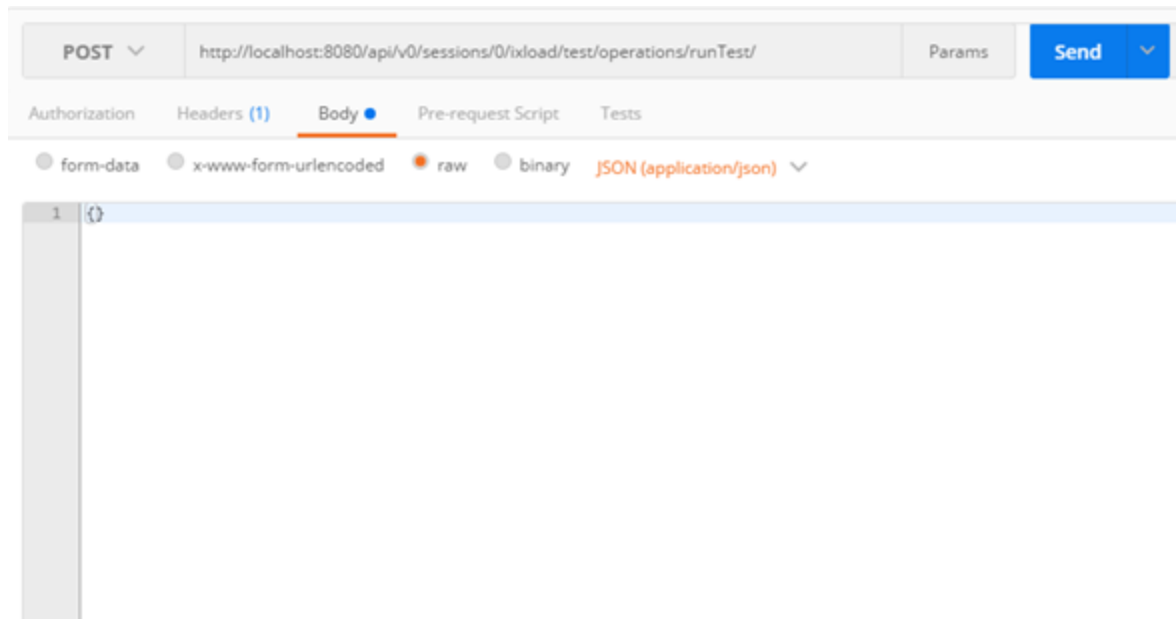
In the payload and body of the request, add the path to the .rxf file:

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

Example of running a test

On an active 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 of waiting to capture the port data

On an active session, do a POST on a URL similar to the following:

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

Example of stopping a test

On an active 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
```

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

- Method:** POST
- URL:** `http://localhost:8080/api/v0/sessions/0/ixload/test/operations/abortAndReleaseConfigWaitFinish`
- Params:** (empty)
- Body:** `{}` (JSON format)
- Headers:** (1 header)
- Pre-request Script:** (empty)
- Tests:** (empty)

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. 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. For example, this means that a GET request on `/configuredStats?filter="caption eq HTTP"` returns all statistics whose caption contains `HTTP`. If you want a `matches` operation instead, 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.

You can include multiple query string conditions in the same URL by separating them with 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 that satisfy the query string conditions.
PATCH	Modifies the parameter list sent in the request payload with all the elements of the list that satisfy the query string conditions.
DELETE	Deletes every element in the list that satisfies the query string conditions.

Collecting diagnostics

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

To collect diagnostics, ensure the following:

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

To collect diagnostics, use the following command:

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

Specify the ZIP file location as the POST payload:

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

For example:

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

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

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

☐ GET ☒ POST ☐ PUT ☐ DELETE Other methods ▼ application/json ▼

Raw headers Headers form Headers sets

Content-Type: application/json

Raw payload Data form Files (0)

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

SEND

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

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

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

Raw JSON

`{}`

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

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

For example:

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

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

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

GET POST PUT DELETE Other methods

Raw headers Headers form Headers sets

Content-Type: application/json

SEND

Status: 200: OK ? Loading time: 25 ms

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

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

Raw JSON

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

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.

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

☐ GET
 ☒ POST
 ☐ PUT
 ☐ DELETE
 Other methods
 application/json

Raw headers Headers form Headers sets

Content-Type: application/json

Raw payload Data form Files (0)

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

SEND

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

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

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

Raw JSON

```
{ }
```

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

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

For example:

Operations

GET @

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

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

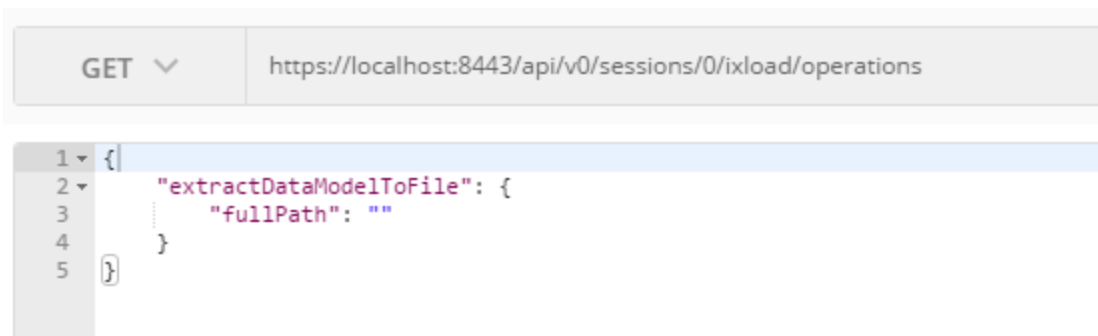
The screenshot displays a REST client interface with the following details:

- URL:** `http://127.0.0.1:8080/api/v0/sessions/72/ixload/test/activeTest/operations/collectDiagnostics/3`
- Method:** GET (selected), POST, PUT, DELETE, Other methods
- Raw headers:** Content-Type: application/json
- Status:** 200: OK (with a help icon) Loading time: 25 ms
- Response headers (4):**
 - Date: Mon, 01 Aug 2016 09:03:59 GMT
 - Content-Length: 118
 - Content-Type: application/json
 - Server: CherryPy/3.6.0
- Request headers (1):**
- Redirects (0):**
- Timings:**
- JSON body:**

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

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" }`).



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
100     - value: False
101     - readOnly: False
102
103 Current resource: /ixload/sessionOverview
104 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
```

findURLs operation

`findURLs` enables you to find the URLs where certain resources can be found. The resources you can find are:

- Property names
- Property values
- URL contents

To use `findURLs` you must have an IxLoad REST API session open with a configuration already loaded inside it.

`findURLs` is available on the `ixload` resource. To execute this operation, make a POST request on an active IxLoad REST session, on the following URL:

<https://IP:8443/api/v0/sessions/X/ixload/operations/findURLs>

The body of the POST request must contain at least one of the following parameters:

- `propertyName`
- `propertyValue`
- `urlContains`

You can pass multiple parameters to `findURLs`. If you do, it returns all the resources that match all the passed parameters. For example, if you send `{"propertyName": "count", propertyValue=100}`, IxLoad returns all the resources in the IxLoad session that contain a field named `count` that have a value of 100.



propertyName

Searching by `propertyName` returns all the URLs that are available under the IxLoad session that contain a property with the provided name. The results of the operation contain all the URLs that satisfy this query, along with the value that property has for each URL.

The example below show all URLs that contain a property called `commandType`, which can be used to find all L47 activity commands:

The screenshot shows the Ixia API Browser interface. The browser address bar displays `https://localhost:8443/#/rest/api/v1/sessions/0/ixload?view=instance`. The Ixia logo and "API Browser" text are in the top left, and "v1.0.8.127" is in the top right. A sidebar on the left lists navigation items: `IXLOAD`, `chassisBuilder`, `chassischain`, `preferences`, `stats`, `test`, and `versions`. The main content area is titled "Method - findURLs" and displays the "Operation Result" for a GET request to `/api/v1/sessions/0/ixload/operations/findURLs/1`. The response is a JSON object indicating success and listing three found URLs with their respective command types and resource URLs.

```

1 GET /api/v1/sessions/0/ixload/operations/findURLs/1
2 200 OK TTFB:33ms
3 date: Wed, 24 Apr 2019 16:19:15 GMT
4 server: CherryPy/unknown
5 access-control-allow-methods: GET,POST,PATCH,DELETE,OPTIONS
6 content-type: application/json
7 access-control-allow-origin: *
8 access-control-expose-headers: Location
9 access-control-allow-headers: Content-Type,Content-Length
10 content-length: 817
11
12 {
13   "foundResult": [
14     {
15       "propertyName": "commandType",
16       "resourceUrl": "/ixload/test/activeTest/communityList/0/activityList/0/agent/actionList/0",
17       "propertyValue": "START"
18     },
19     {
20       "propertyName": "commandType",
21       "resourceUrl": "/ixload/test/activeTest/communityList/0/activityList/0/agent/actionList/2",
22       "propertyValue": "GET"
23     },
24     {
25       "propertyName": "commandType",
26       "resourceUrl": "/ixload/test/activeTest/communityList/0/activityList/0/agent/actionList/1",
27       "propertyValue": "STOP"
28     }
29   ],
30   "url": "/api/v1/sessions/0/ixload/operations/findURLs/1",
31   "state": "SUCCESS",
32   "result": "",
33   "progress": 100,
34   "type": "findURLs",
35   "id": 1
36 }

```

propertyValue

Searching by `propertyValue` returns all the resources in the currently open IxLoad REST session datamodel that contain a property that has the provided value. The results of the operation contain all the URLs that satisfy this condition, along with the name of the property that has that value.

The image below shows the results of a `findURLs` query to find all the URLs in the loaded rxf that have a value of 100. The results in the image show that `findURLs` found that components such as `rampUp`, `objectiveValue`, and `ipCount`, and others to have the searched-for value of 100.

You can use `propertyValue` to find boolean, integer, and string properties.

The screenshot shows the IXIA API Browser interface. The address bar displays the URL: `https://localhost:8443/#/rest/api/v1/sessions/0/ixload?view=instance`. The left sidebar contains a tree view with the following items: `ixload`, `chassisBuilder`, `chassischain`, `preferences`, `stats`, `test`, and `versions`. The main panel is titled "Method - findURLs" and displays the "Operation Result" for a GET request to `/api/v1/sessions/0/ixload/operations/findURLs/2`. The response is a 200 OK status with a content type of `application/json`. The JSON body contains an array of results, each with a `propertyName`, `resourceurl`, and `propertyvalue`.

```

1 GET /api/v1/sessions/0/ixload/operations/findURLs/2
2 200 OK TTFB:27ms
3 date: Wed, 24 Apr 2019 16:21:14 GMT
4 server: CherryPy/unknown
5 access-control-allow-methods: GET,POST,PATCH,DELETE,OPTIONS
6 content-type: application/json
7 access-control-allow-origin: *
8 access-control-expose-headers: Location
9 access-control-allow-headers: Content-Type,Content-Length
10 content-length: 3966
11
12 {
13   "foundResult": [
14     {
15       "propertyName": "logCollectorsSize",
16       "resourceurl": "/ixload/preferences",
17       "propertyvalue": "100"
18     },
19     {
20       "propertyName": "rampupTime",
21       "resourceurl": "/ixload/test/activeTest/timelineList/0",
22       "propertyvalue": "100"
23     },
24     {
25       "propertyName": "totalUserObjectiveValue",
26       "resourceurl": "/ixload/test/activeTest/communityList/0",
27       "propertyvalue": "100"
28     },
29     {
30       "propertyName": "timerGranularity",
31       "resourceurl": "/ixload/test/activeTest/communityList/0/activityList/0",
32       "propertyvalue": "100"
33     },
34     {
35       "propertyName": "constraintValue",
36       "resourceurl": "/ixload/test/activeTest/communityList/0/activityList/0",
37       "propertyvalue": "100"
38     },
39     {
40       "propertyName": "secondaryConstraintValue",
41       "resourceurl": "/ixload/test/activeTest/communityList/0/activityList/0",
42       "propertyvalue": "100"
43     },
44     {
45       "propertyName": "userObjectiveValue",
46       "resourceurl": "/ixload/test/activeTest/communityList/0/activityList/0",
47       "propertyvalue": "100"
48     },
49     {
50       "propertyName": "propertyvalue"

```

urlContains

Searching by `urlContains` finds all the resources whose URL contains the string provided as a parameter. For example, the image below shows the results of searching for `vlanRange`:

Operation Result

IxLoad Session Handling

Creating and handling IxLoad sessions is done through IxLoadGateway, which is an IxLoad service. IxLoadGateway is installed with IxLoad as part of the custom install options (see [Before you Begin on page xiv](#)).

Creating a new session

There three ways to create a session:

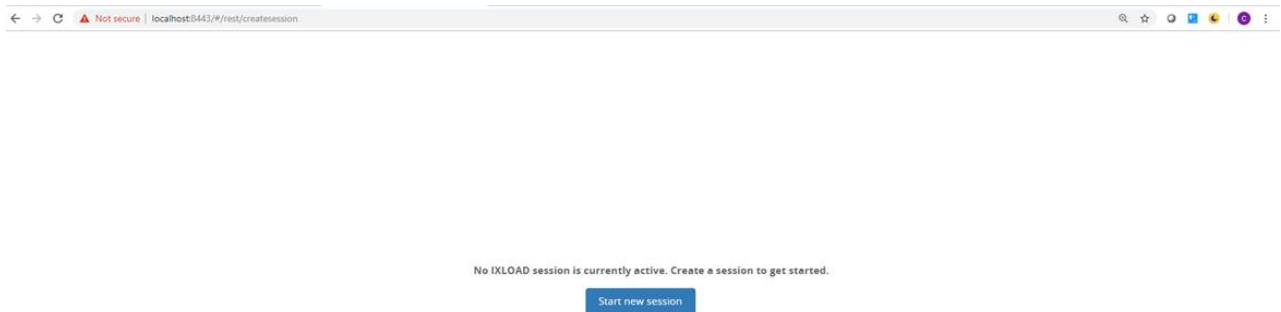
- By specifying the version of IxLoad to use to create the session.
To specify the version to use, perform a POST on `/sessions` with one of the following payloads, appropriate for the URL type you are using:

- `/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
- By connecting to the [API Browser](#) (<https://localhost:8443/>) while no IxLoad REST sessions are active.

If you connect to the API Browser while no IxLoad REST sessions are active, the page shown below displays, which enables you to start a new session.



New session with a specified version

To create a new session with a specific version of IxLoad, do a POST on `api/v0/sessions` with a payload of `{"ixLoadVersion":"version no."}`.

IxLoad Session Handling

This action creates a session, but does not start it or make it active. This action does not take into consideration the instance count limit on the client side. The instance count limit is only considered when sessions are started.

The following figure shows an example of starting a new session with a specific IxLoad version in a REST client:

The screenshot shows a REST client interface. The URL bar contains `http://127.0.0.1:8080/api/v0/sessions/`. Below the URL bar, there are radio buttons for HTTP methods: GET, POST (selected), PUT, PATCH, DELETE, HEAD, OPTIONS, and Other. Below the methods, there are tabs for Raw, Form, and Headers. The Headers tab is active, showing a JSON payload: `{"ixLoadVersion": "8.00.0.195"}`. The payload is displayed in a text area with a "Raw" tab selected. Below the text area, there are buttons for "Encode payload" and "Decode payload".

The following figure shows the response for the POST request in the preceding figure. Note that the status is 201 Created and Location points to the new session.

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

Starting a session

To start a specified-version session, you use the `start` operation. This operation starts a new IxLoad session based on the IxLoad version for which the session was created.

`start` is available on each individual session and requires no payload. The following figure shows how a `start` operation appears in the REST client:

The screenshot shows the IxLoad web interface. At the top, there is a URL bar with the address `http://127.0.0.1:8080/api/v0/sessions/1/operations/start/`. Below the URL bar, there are radio buttons for selecting the HTTP method: GET, POST (selected), PUT, PATCH, DELETE, HEAD, OPTIONS, and Other. Underneath the method selection, there are three tabs: Raw, Form, and Headers. The Headers tab is currently selected. Below the tabs, there are two buttons: 'Encode payload' and 'Decode payload'. The main area below these buttons is empty.

The following figure shows the response for the `start` operation. The response is `202 Accepted` and `Location` shows the result for the operation.

Status	202 Accepted Loading time: 14 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36 Origin: chrome-extension://hgmloofddfnphfcgellkdfbfbjeloo Content-Type: application/json Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.8 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
Response headers	Date: Wed, 07 Oct 2015 14:43:41 GMT Content-Length: 2 Content-Type: application/json Location: api/v0/sessions/1/operations/start/1 Server: CherryPy/3.6.0

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

IxLoad Session Handling

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

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form Headers

Clear Send

Status **200 OK** Loading time: 8 ms

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

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

Raw JSON Response

Copy to clipboard Save as file

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

The following figure shows an example a `start` operation that failed because the maximum number of instances was already active:

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

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

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

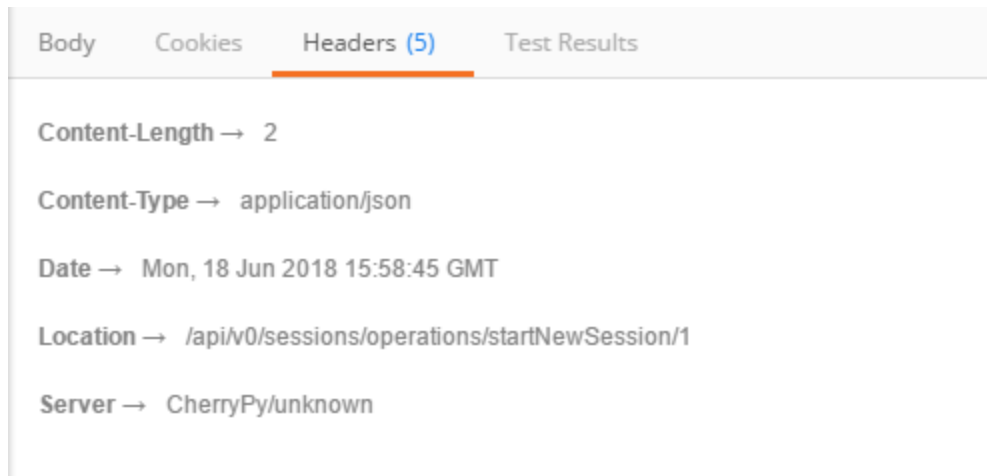
```
{
  status: "Successful"
  actionName: "start"
  state: "finished"
  errorMessage: "Already running maximum allowed copies of IxLoad."
  sessionId: 2
}
```

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 `startNewSession` as shown in the following URL:

`http://localhost:8080/api/v0/sessions/operations/startNewSession`

`startNewSession` does not require a payload. As for every REST API operation, the headers of the response contain a `Location` URL you can use to check the status of the `startNewSession` operation:



After the `startNewSession` operation finishes, the `status` URL will display the application version that was used, along with the ID of the session that was created:



Deleting a session

You delete an IxLoad session in the same way as for generic lists: you send a `DELETE` request either to the sessions list URL, or to the specific session's object ID.

- If you send the request to the sessions URL, all sessions will be closed.
- If you send the request to a specific session's object ID, only that session will be closed.

When deleting a session, the IxLoad process underneath it will also 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, do 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`:

Key	Value
<input checked="" type="checkbox"/> Content-Type	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:

form-data x-www-form-urlencoded raw **binary**

Choose Files demo.rxf

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.

Uploading and downloading files

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:

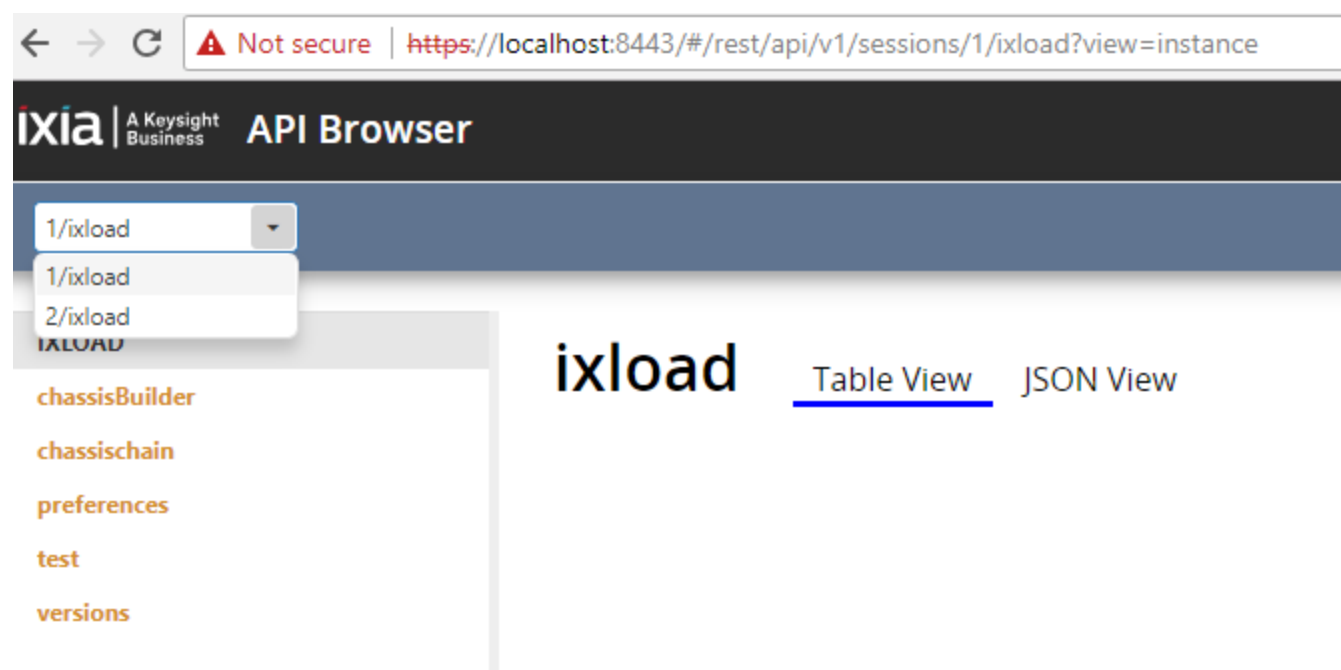
`https://localhost:8443/`

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)

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.

command
Table View JSON View

Edit
Operations
History

Attribute	Value	Description
abort	None	rw

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

Table View JSON View			Add	Remove	
<input type="checkbox"/>	Instance	protocolAndType			
<input type="checkbox"/>	activityList/0	HTTP Client			

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

test Table View JSON View Edit Operations

Attribute

expirationTimer

loadedRxf

outputDir

responseObjectType

runResultDirFull

REST Operations

OPERATIONS

`abortandreleaseconfigwaitfinish` This operation will deconfigure a configured IxLoad Test.
`abortandreleaseconfigwaitfinish()`

`applyconfiguration` This operation will run an apply configuration on the current IxLoad Test. The state of the test will be Configured if successful.
`applyconfiguration()`

`exportconfig` This operation exports the current configuration to a requested location as a compressed repository file.
`exportconfig(string)`

`gracefulstoprun` This is the operation used to gracefully stop an IxLoad Test.
`gracefulstoprun()`

`importconfig` This operation imports the provided compressed repository file and saves the repository file to the specified location.
`importconfig(string,string)`

`loadtest` This operation will load the provided repository
`loadtest(string)`

`runtest` This is the operation used to start an IxLoad Test.
`runtest()`

`save` This operation will save the current repository to the default result location.
`save()`

`saveas` This operation will save the current repository to specific location.
`saveas(string,boolean (true|false))`

`waitforallcapturedata` This is the operation used after a test runs, to wait until all capture data was received.
`waitforallcapturedata()`

How to find URLs in a REST API session

Note:

There two methods to find URLs in a REST API session:

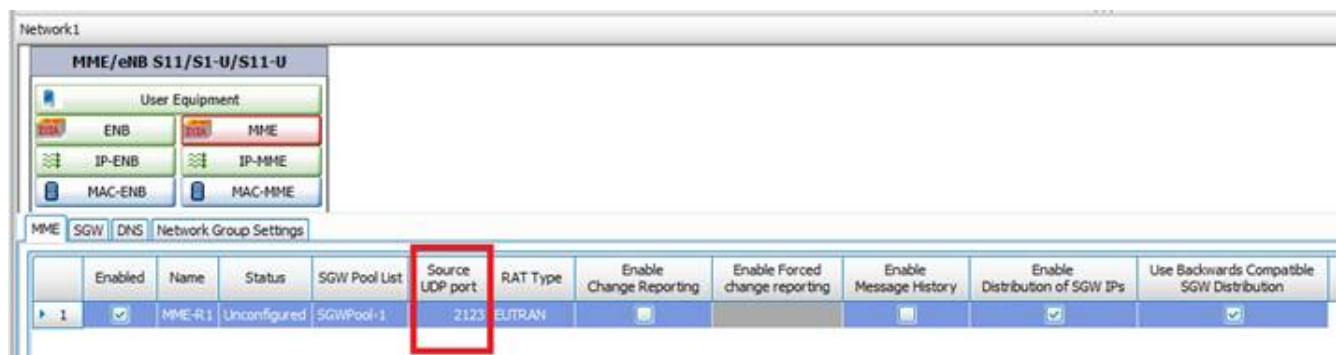
- The `extractDataModelToFile` method, described in this section.
- The `findURLs` method, described in [findURLs operation on page 36](#).

`extractDataModelToFile` is the original method. `findURLs` is a newer, more efficient method.

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:

MME-R1 Table View JSON View

Attribute	Value	Description
enableEchoRequest	false	rw
enableForceChangeReporting	false	r
enableMsgHistory	false	rw
enableSgwDistribution	true	rw
ipRange		rw
itemType	EGTPMMERange	rw
modifyBearerCommandQ	5	rw
modifyBearerCommandT3	3	rw
modifyBearerN3	5	rw
modifyBearerT3	3	rw
name	MME-R1	rw
ratType	6	rw
releaseBearerN3	5	rw
releaseBearerT3	3	rw
restObjectType	inNetEGTPMMERange	r
s1uDataTransferIsSupported	true	rw
setMABR	false	rw
setNTSR	false	rw
setPRN	false	rw
sgwCount	1	rw
sgwVp	20.0.0.1	rw
srcUdpPort	2123	rw
useBackwardsCompatibleSgwDistribution	true	rw
usePlaneCotEpsOptimisationIsSupported	false	rw

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

You can use the REST API to browse the IxLoad data model to retrieve or modify the current configuration. This section describes where to find resources such as L4-7 plugins, L2-3 ranges, and timelines in the data model. In addition, it describes operations such as loading and saving configurations and running a test.

Communities

You can find all the communities on 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, or peer. In addition, this list 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 to filter the clients:

```
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 are in the `timelineList`, 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 (that is, the activities do not have a `timeline` option exposed). Instead, 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 used by a running session by changing the `loginName` field on the `chassischain` resource:

Perform a PATCH on `http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/`

with a payload of: { "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 `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.

`expirationTimer` is exposed as a field under a URL as follows:

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

To modify the value of this field, 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 `expirationTimer` can be updated to a new value at any time. The update resets the timer to a new value, meaning that the session will be deleted after the new timer has expired.

To cancel the 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 the 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

To add a chassis perform a POST as follows:

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

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

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

- URL:** `http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList`
- Method:** ☒ GET ☒ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other
- Tab:** Raw (selected), Form, Headers
- Payload:** `{"name":"10.215.170.77"}`
- Content-Type:** `application/json` (selected) with a dropdown arrow and the text "Set 'Content-Type' header to overwrite this value."
- Buttons:** Clear, Send

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

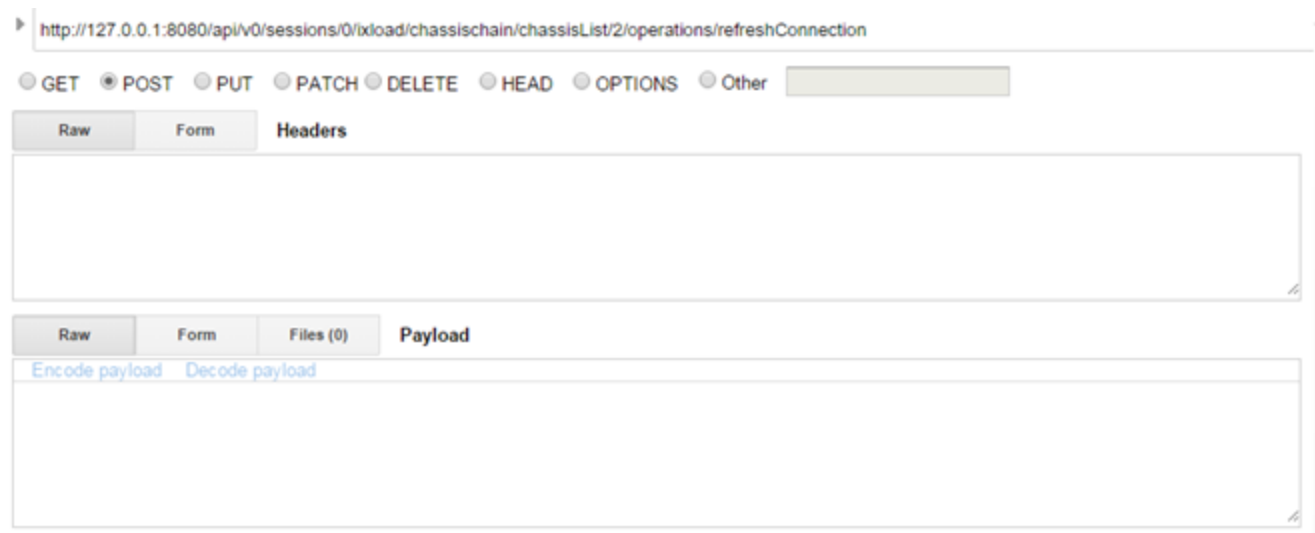
Status	201 Created Loading time: 12 ms
Request headers	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36 Origin: chrome-extension://hgmlcoofddfdnphfgcellkdfbfjeloo Content-Type: application/json Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.8 Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
Response headers	Date: Wed, 07 Oct 2015 14:52:11 GMT Content-Length: 2 Content-Type: application/json Location: /api/v0/sessions/0/ixload/chassischain/chassisList/2 Server: CherryPy/3.6.0

Connecting to a chassis

To connect to a chassis, perform a POST as follows:

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

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



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

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

The result of the refresh operation is as follows:

http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList/2/operations/refreshConnection/0

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Status: 200 OK Loading time: 16 ms

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

Response headers: Date: Wed, 07 Oct 2015 15:03:18 GMT
Content-Length: 138
Content-Type: application/json
Server: CherryPy/3.6.0

Raw JSON Response

Copy to clipboard Save as file

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

Note that there is a new field inserted that is named `refreshedChassis`. This refers to the IP or hostname of the chassis that was refreshed.

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

To handle cases in which an rxf contains a chassis that no longer exists, a warning field in the `refreshConnection` operation indicates that a chassis is missing and the `refreshedChassis` field contains only those chassis that were successfully connected to. The figure below shows an example of this: a GET on the status of the `refreshConnection` operation shows that no chassis were refreshed and a warning message displays, describing the error.

The screenshot displays a web browser's developer console with the following details:

- Status:** 200 OK (blue checkmark icon) Loading time: 16 ms
- Request headers:**
 - User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
 - Content-Type: text/plain; charset=utf-8
 - Accept: */*
 - Accept-Encoding: gzip, deflate, sdch
 - Accept-Language: en-US,en;q=0.8
 - Cookie: JSESSIONID=6F4A7F28464E06921C8784F490B8A464
- Response headers:**
 - Date: Wed, 07 Oct 2015 15:09:06 GMT
 - Content-Length: 297
 - Content-Type: application/json
 - Server: CherryPy/3.6.0

Below the headers, the **Response** tab is selected, showing the following JSON data:

```
{
  status: "Successful"
  actionName: "refreshConnection"
  state: "finished"
  warning: "Could not connect to chassis 10.205.29.21. If any ports were assigned to the network they have been removed. Please reassign if chassis will be back up."
  refreshedChassis: ""
}
```

Removing a chassis

To remove a chassis, you perform 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 specific chassis, the DELETE request must be performed on the following URL:

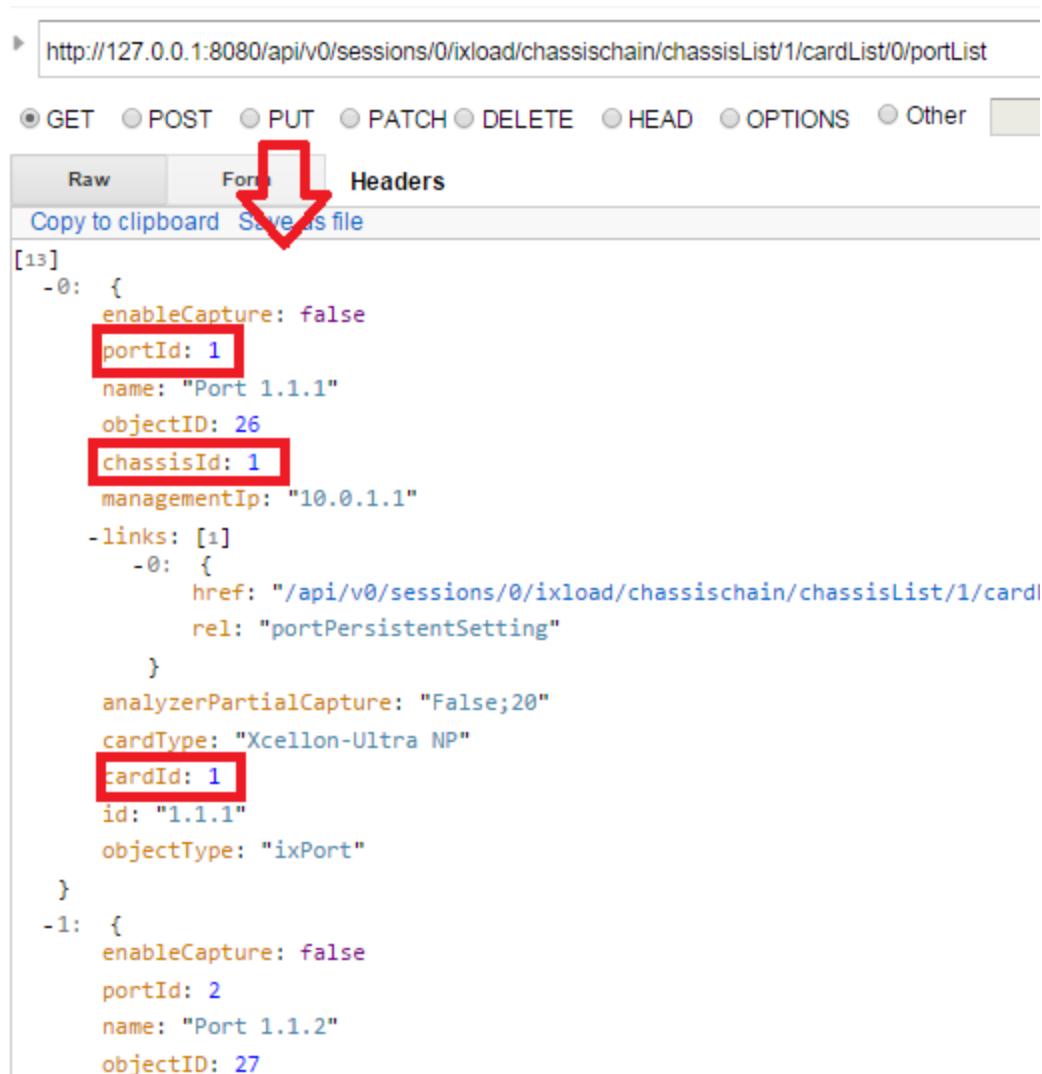
```
api/v0/sessions/0/ixload/chassischain/chassisList/chassisObjectId
```

Removing a chassis is similar to DELETE operations on other IxLoad Data Model lists.

Assigning ports

To assign ports, you perform a POST operation on the network port list. The POST request requires three parameters: `chassisId`, `cardId`, and `portId`. These parameters do not represent the unique objectIDs used by the REST API to identify resources as part of a list. Instead, these three parameters have the same meaning they have in the UI and TCL/Python/Perl scripting, where a port is identified by a string such as 1.1.1 (`chassis.card.port`).

To obtain the `chassisId`, `cardId`, and `portId`, perform a GET request on the `portList` for each card in a chassis, as shown in the following figure:



http://127.0.0.1:8080/api/v0/sessions/0/ixload/chassischain/chassisList/1/cardList/0/portList

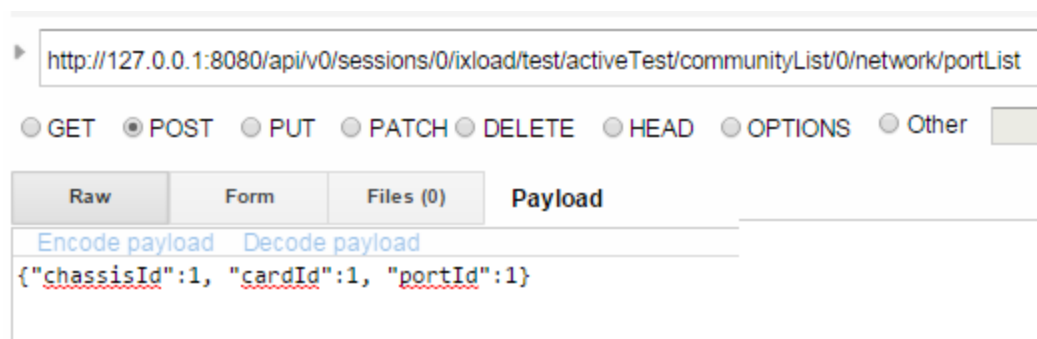
GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Headers

Copy to clipboard Save as file

```
[13]
-0: {
  enableCapture: false
  portId: 1
  name: "Port 1.1.1"
  objectID: 26
  chassisId: 1
  managementIp: "10.0.1.1"
  -links: [1]
    -0: {
      href: "/api/v0/sessions/0/ixload/chassischain/chassisList/1/card
      rel: "portPersistentSetting"
    }
  analyzerPartialCapture: "False;20"
  cardType: "Xcellon-Ultra NP"
  cardId: 1
  id: "1.1.1"
  objectType: "ixPort"
}
-1: {
  enableCapture: false
  portId: 2
  name: "Port 1.1.2"
  objectID: 27
```

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



http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/0/network/portList

GET POST PUT PATCH DELETE HEAD OPTIONS Other

Raw Form Files (0) Payload

Encode payload Decode payload

```
{\"chassisId\":1, \"cardId\":1, \"portId\":1}
```

Taking or clearing ownership of ports

To take or clear ownership of ports, you perform POST requests on the port object:

Take ownership

```
api/v0/sessions/0/ixload/chassischain/chassisList/chassis_ID/cardList/card_ID/portList/port_ID/operations/takeOwnership
```

Clear ownership

```
api/v0/sessions/0/ixload/chassischain/chassisList/chassis_ID/cardList/card_ID/portList/port_ID/operations/clearOwnership
```

If another user owns the port, you can forcefully clear their ownership of the port by setting the `force` parameter to `true` in the body of the request. For example: `{"force": "true"}`.

Rebooting ports

To reboot ports:

1. Call the `refreshConnection` operation on the card.
2. Reboot the ports by performing a POST operation on the port object:

```
api/v0/sessions/0/ixload/chassischain/chassisList/chassis_ID/cardList/card_ID/portList/port_ID/operations/reboot
```

Unassigning ports

To unassign ports, you perform 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

Chassis Chain/Port Assignment Operations

```
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}
```

The screenshot shows a web browser interface for an HTTP client. The address bar contains the URL: `http://localhost:8080/api/v0/sessions/0/ixload/chassisBuilder/operations/getChassisSettings/0/`. Below the address bar, the HTTP method is set to GET. The response status is 200 OK, and the loading time is 312ms. The response headers are displayed, including Date, Content-Length, Content-Type, and Server. The response body is shown in JSON format, indicating a successful operation with a link to the result.

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

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

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

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

The result is in the following form:

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

This page intentionally left blank.

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:

The image contains two screenshots of a REST client interface. The top screenshot shows the 'Headers' tab with a table where the 'Content-Type' header is set to 'multipart/form-data'. The bottom screenshot shows the 'Body' tab with the 'binary' radio button selected and a file named 'hss_cx.xml' attached.

KEY	VALUE
<input checked="" type="checkbox"/> Content-Type	multipart/form-data
Key	Value

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw **binary**

Choose Files hss_cx.xml

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, nettraffic, and activity.

Your test must poll statistics from the web server. The web server holds all the statistics configured in the test in a circular buffer for a default amount of polls of 20 timestamps. 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.

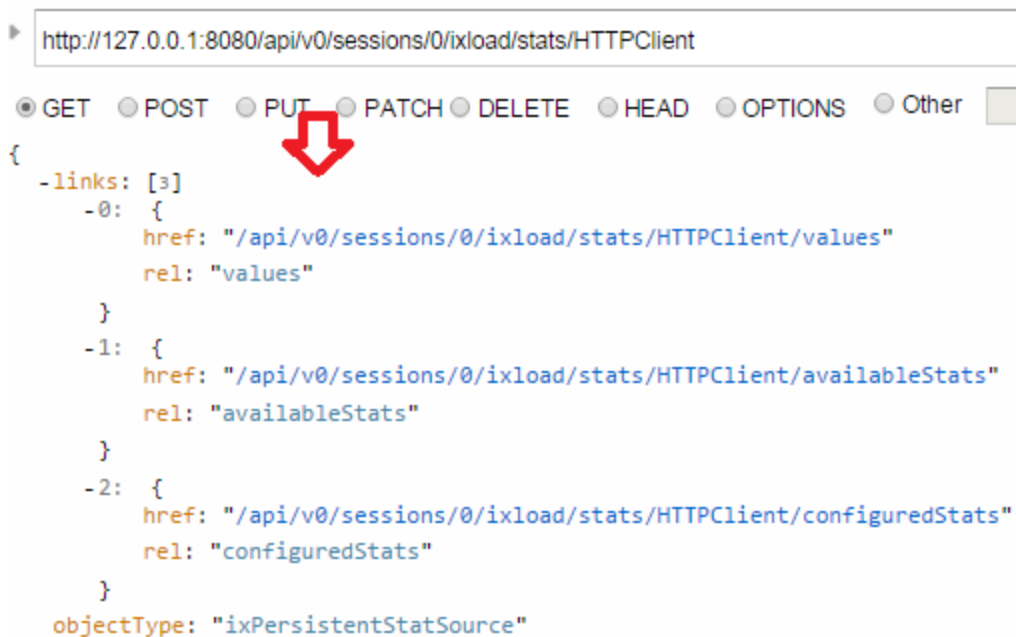
The root resource for statistics is the following URL:

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

To retrieve the list of statistics sources, perform a GET request on this URL, as shown in the following figure:



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:



`availableStats` is a list of all the available statistics for the current test. This list is read-only; you cannot remove the available statistics.

`configuredStats` is a list of 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 available statistics (that is, it contains the `availableStats` list).

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`

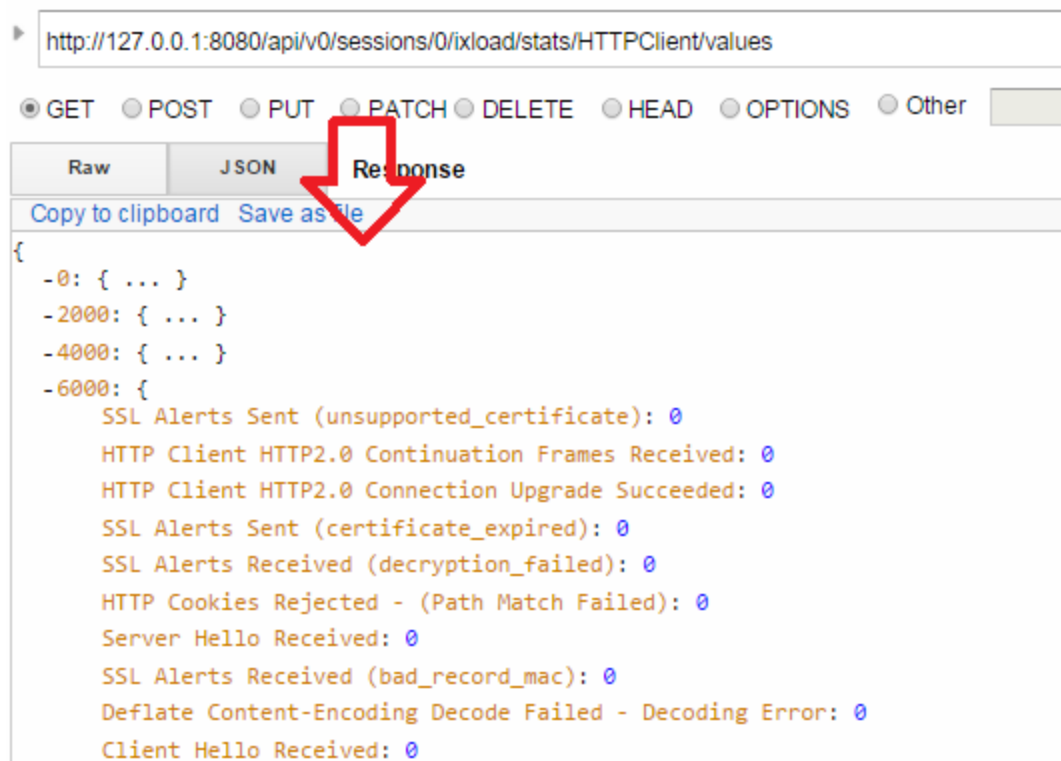
`values` is a dictionary that contains the actual statistics values during the IxLoad test run.

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 only retains the last 20 timestamps. If you do not poll the statistics frequently enough, you might lose some timestamps.

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



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:

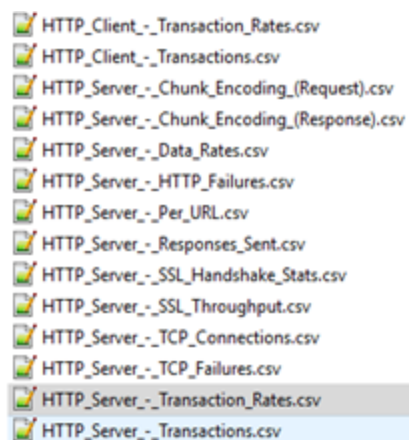
<http://127.0.0.1:8080/api/v0/sessions/sessionId/ixload/stats/restStatViews/statViewId/statList>

```
[
  {
    "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"
  }
]
```

Enabling 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>



- HTTP_Client_-_Transaction_Rates.csv
- HTTP_Client_-_Transactions.csv
- HTTP_Server_-_Chunk_Encoding_(Request).csv
- HTTP_Server_-_Chunk_Encoding_(Response).csv
- HTTP_Server_-_Data_Rates.csv
- HTTP_Server_-_HTTP_Failures.csv
- HTTP_Server_-_Per_URL.csv
- HTTP_Server_-_Responses_Sent.csv
- HTTP_Server_-_SSL_Handshake_Stats.csv
- HTTP_Server_-_SSL_Throughput.csv
- HTTP_Server_-_TCP_Connections.csv
- HTTP_Server_-_TCP_Failures.csv
- HTTP_Server_-_Transaction_Rates.csv
- HTTP_Server_-_Transactions.csv

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 `RunState`. 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"
},

```


[illegible]

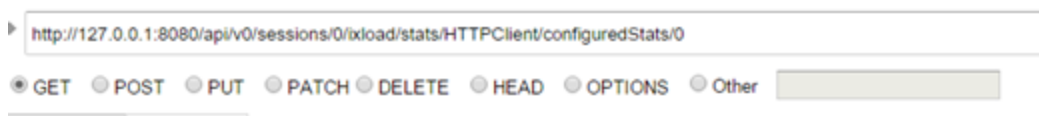
- **kSum** – there is one value for all users (for example, the **Stream Bit Rate** statistic):

```
{
  "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

To change statistics, you perform a PATCH method on the configured statistics structure. You can turn statistics on or off, or change the aggregation type.

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



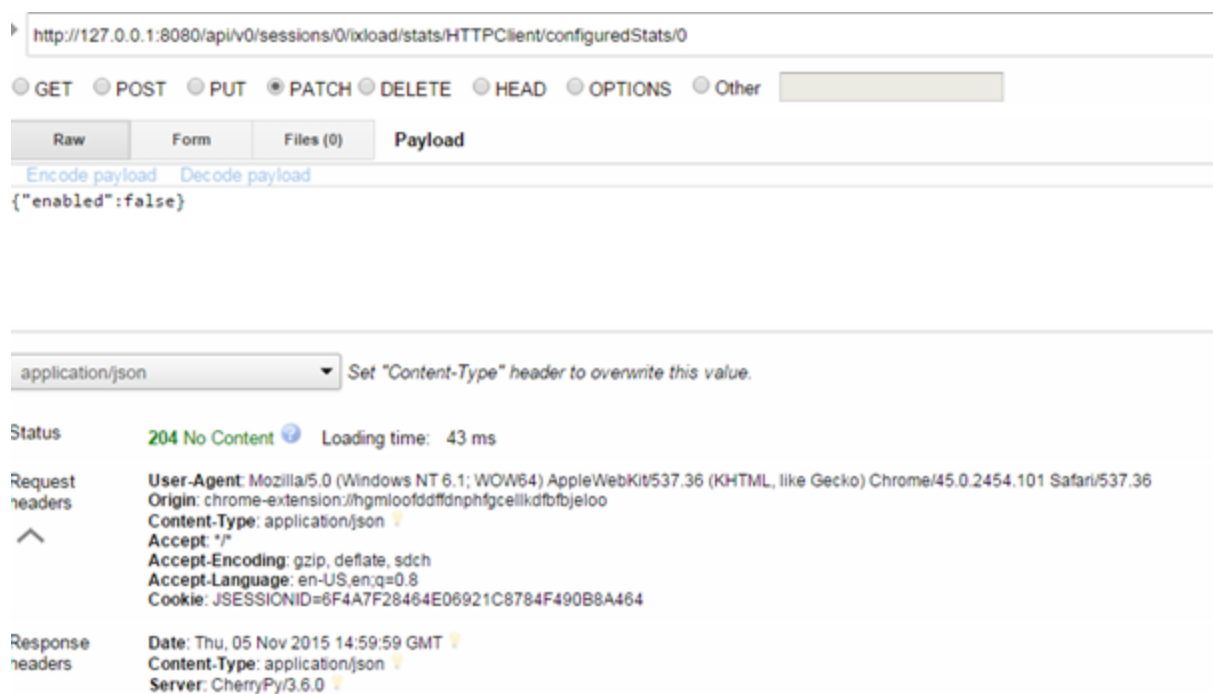
The result of GET in the preceding request is shown in the following figure:



The screenshot shows a REST client interface with three tabs: 'Raw', 'JSON', and 'Response'. The 'JSON' tab is selected, displaying a JSON object. Above the JSON, there are links for 'Copy to clipboard' and 'Save as file'. The JSON object contains a list of links, a boolean 'enabled' set to true, and several descriptive fields for the 'HTTP Simulated Users' statistic.

```
{
  -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 to be changed.



The screenshot shows a REST client interface for a PATCH request. The URL is `http://127.0.0.1:8080/api/v0/sessions/0/ixload/stats/HTTPClient/configuredStats/0`. The method is set to PATCH. The payload is `{"enabled": false}`. The status is 204 No Content, and the response headers show the date and content type.

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

Method: ☐ GET ☐ POST ☐ PUT ☒ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form Files (0) Payload

Encode payload Decode payload

Payload: `{"enabled": false}`

Content-Type: `application/json` Set "Content-Type" header to overwrite this value.

Status: **204 No Content** Loading time: 43 ms

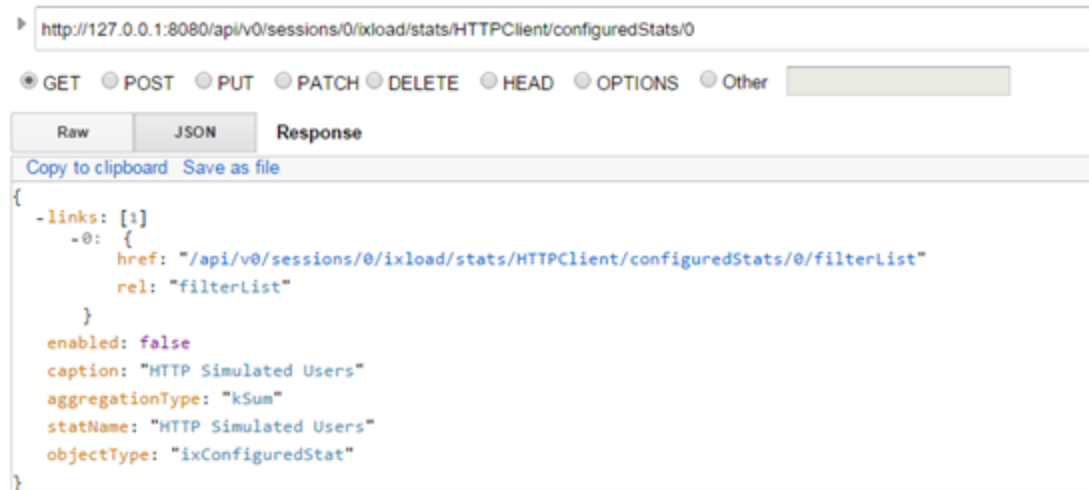
Request headers:

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

Response headers:

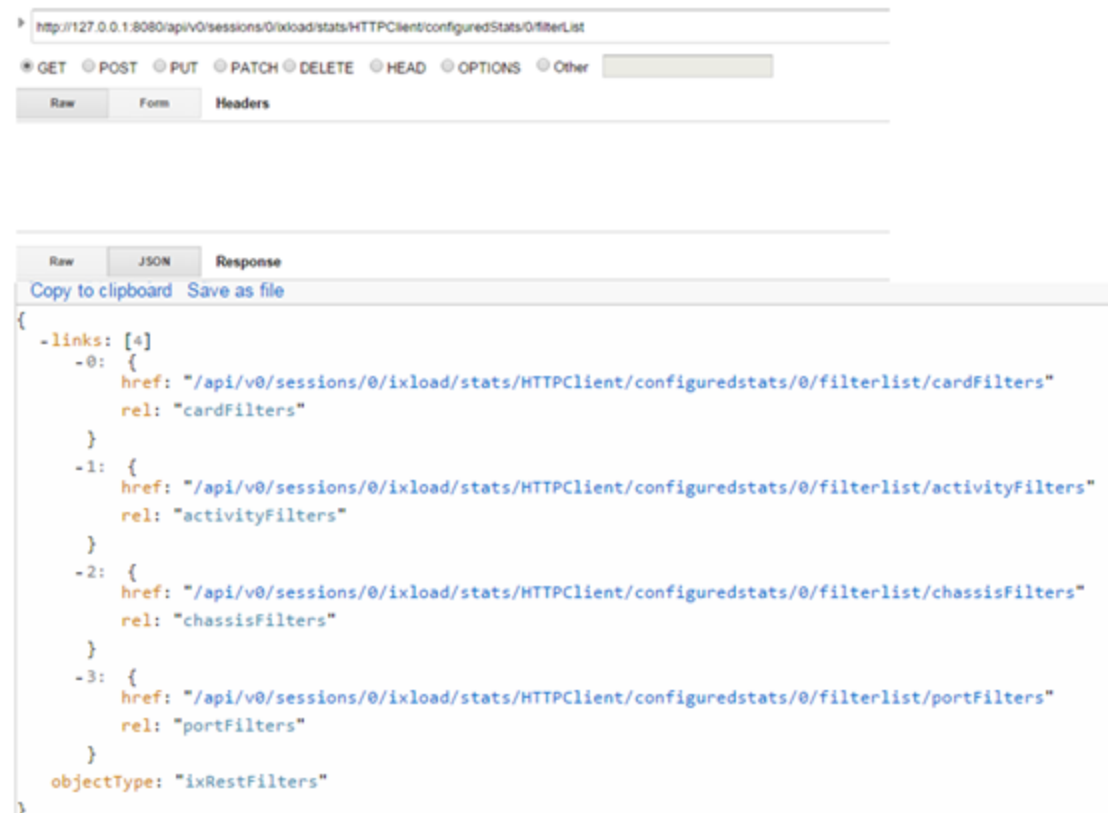
- Date: Thu, 05 Nov 2015 14:59:59 GMT
- Content-Type: application/json
- Server: CherryPy/3.6.0

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



Filtering stats

To obtain the filtered statistics, you perform a GET on the filter list from a specific `configuredStat` item, as shown in the following figure:



A configured statistic contains filters that enable you to get values at various levels:

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

To add a port filter, you add a new port to the `portFilter` list, as shown in the following figure:

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

☐ GET ☒ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form Headers

Raw Form Files (0) Payload

Encode payload Decode payload

```
{ "value": "10.215.170.45/Card1/Port1" }
```

application/json Set "Content-Type" header to overwrite this value.

Clear Send

Status **201 Created** Loading time: 47 ms

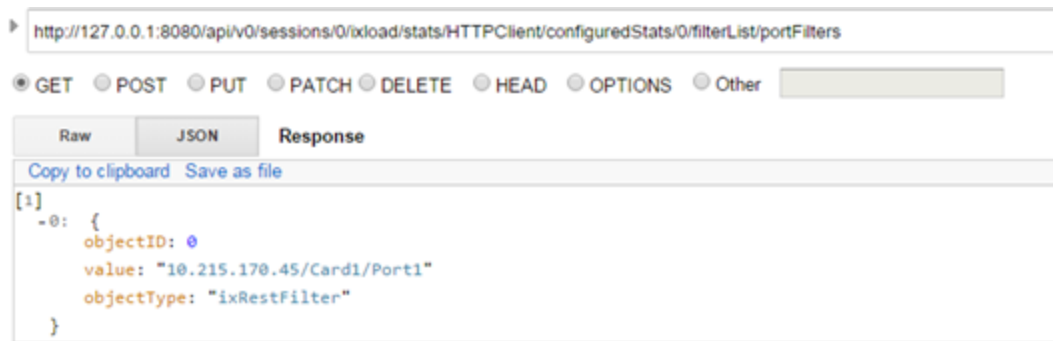
Request headers

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

Response headers

Date: Thu, 05 Nov 2015 15:04:12 GMT
Content-Length: 2
Content-Type: application/json
Location: /api/v0/sessions/0/xdload/stats/HTTPClient/configuredStats/0/filterList/portFilters/0
Server: CherryPy/3.6.0

The following figure shows how the filter looks after it has been added:



You can set multiple filters for multiple configured statistics according to how you want to view the statistics. Aggregation and processing can be done in the client script after the statistics are coming in.

Adding an activity filter

To add activity filter to a statistic, you perform 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 on a custom path, use the following operation on the `test` resource before running the configuration:

Perform a PATCH on `http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/`

with the following payload:

```
outputDir : true (the default is 'false')
runResultDirFull : "F:\\path\\to\\the\\new\\result\\dir"
```

Logging

You can retrieve log from the REST API that are 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 19:03:35.836",
    "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 named `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

To delete the logs for a instance, perform a PATCH operation on the session URL with a payload of `{"deleteLogsOnSessionClose" : true}`. This will cause all the session logs (the `IxLoadRest-xyy.log` and all the client logs for the IxLoad instance used by the session) to be deleted when the session is deleted (using the DELETE operation).



Deleting logs for a specific IxLoad version

To delete the logs for a specific IxLoad version, perform a POST operation on the following URL:

`@api/v0/logs/operations/deleteVersionLogs`

with a payload of:

```
{"appVersion" : IxLoad-version}
```

and a header of:

```
{"content-type": "application/json"}
```

This will delete all the logs (all `IxLoadRest-x-yy.log` files and all client logs resulting from all the sessions that used the specified version of IxLoad) for the specified IxLoad version.

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

- Method:** POST
- URL:** `http://localhost:8080/api/v0/logs/operations/deleteVersionLogs`
- Authorization:** (empty)
- Headers (1):** (empty)
- Body:** Selected tab, showing a JSON payload: `["appVersion" : "8.50.0.166"]`
- Pre-request Script:** (empty)
- Tests:** (empty)
- Content-Type:** JSON (application/json)

Deleting logs for all IxLoad versions

To delete the logs for versions of IxLoad installed, perform a POST operation on the following URL:

`@api/v0/logs/operations/deleteAllLogs`

with no payload but with the following header:

```
{"content-type": "application/json"}
```

This will delete all the logs (all `IxLoadRest-x-yy.log` files and all client logs) for all IxLoad versions present on the machine.

This page intentionally left blank.

REST Script Templates

An installed IxLoad build contains a set of Python sample scripts that perform basic IxLoad operations from REST.

The scripts are stored in the root installation folder of IxLoad in a subfolder named `RestScripts`.

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

Before you run the scripts, you must change the configuration data (IxLoad Version, chassis, rxf file path) in the beginning of each script to match your configuration.

The REST scripts 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.

These two files are helper files that implement a Python script to handle REST communication with the IxLoad REST framework.

The scripts are intended as guides to using IxLoad from REST. They expose basic workflow scenarios as examples you can use to understand how to automatically configure IxLoad through REST. You do not necessarily need to write your own scripts in Python; the IxLoad REST API is compatible with any programming language that supports running HTTP requests.

AddNewCommand.py

This template does the following:

1. Creates a session
2. Loads an Rxf
3. Clears the chassis list
4. Adds a chassis
5. Assigns ports to the networks
6. Clears the command list for client activity
7. Updates the command list of the client HTTP activity by:

- Adding a GET command with custom properties
 - Adding a POST command with custom properties
8. Saves the Rxf
 9. Starts the test
 10. Polls the stats
 11. Closes the IxLoad session

ChangeAgentObjectives.py

This template does the following:

1. Creates a session
2. Loads an Rxf
3. Clears the chassis list
4. Adds a chassis
5. Assigns ports to the networks
6. Updates the activity options by:
 - Enabling constraints
 - Setting a constraint value
 - Changing the objective type
 - Setting a new objective type
7. Saves the Rxf
8. Starts the test
9. Polls the stats
10. Closes the IxLoad session

ChangeIpType.py

This template does the following:

1. Creates a session
2. Loads an Rxf
3. Clears the chassis list
4. Adds a chassis
5. Assigns ports to the networks
6. Updates the IP ranges by changing the count and the IP address
7. Saves the Rxf
8. Starts the test
9. Polls the stats
10. Closes 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 an 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:

1. Creates a session
2. Loads an Rxf
3. Clears the chassis list
4. Adds a chassis

5. Assigns ports to the networks
6. Saves the Rxf
7. Starts the test
8. Polls the stats
9. Closes 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 utilities:

class Connection(__builtin__.object)

This class executes the HTTP requests to the application instance. It handles creation of the HTTP session and execution of HTTP methods.

Methods

Methods defined in this class are as follows:

```
__init__(self, siteUrl, apiVersion)
```

Arguments:

`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. Returns the HTTP reply.

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

Method for calling HTTP GET. Returns 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. Returns the HTTP reply.

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

Method for calling HTTP POST. Returns the HTTP reply.

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

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.

Arguments:

`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, the default is used.

Class methods

Class methods defined here are as follows:

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

Joins two URLs. If the second URL is absolute, the base is ignored.

Ixia recommends that you use `urljoin` instead of `urlparse.urljoin` for the following reasons:

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

Data descriptors

Data descriptors defined here are as follows:

`__dict__`

Dictionary for instance variables (if defined)

`__weakref__`

List of weak references to the object (if defined)

Other attributes

Data and other attributes defined here are as follows:

`kContentJson` = 'application/json'

`kHeaderContentType` = 'content-type'

class WebList(__builtin__.list)

This class transforms a JSON list into a list of `WebObject` instances.

Methods

Methods defined in this class are:

`__init__(self, entries=[])`

Creates a `WebList` from a list of items that are processed by the `_WebObject` function.

Data descriptors

Data descriptors defined in this class are as follows:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class WebObject(__builtin__.object)

This class sets the fields of a WebObject instance to correspond to the JSON format received in a GET request. For example, a response in the format: {"caption": "http"} returns an object that has `obj.caption="http."`

Methods

Methods defined in this class are as follows:

`__init__(self, **entries)`

Creates a WebObject instance by providing a dictionary having a property - value structure.

`getOptions(self)`

Gets the JSON dictionary which represents the WebObject instance.

Data descriptors

Data descriptors defined in this class 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)`

IxLoadRestUtils

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, etc.

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.

`rxFilePath` 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

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

Syntax: `uploadFile(connection, url, fileName, uploadPath, overWrite)`

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.

