IxLoad

REST API Programming Guide

Version 8.00

November 2015

Copyright and Disclaimer

Copyright © 2015 Ixia. All rights reserved.

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

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

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

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

REST resources 1
Supported features 1
REST API - Supported methods and running operations 2
REST representation 2
Primitive values 2
List Objects 2
REST resources 3
Case conventions 3
IxLoad REST Methods 3
GET 3
PATCH 4
POST 5
DELETE 6
OPTIONS 7
OPERATIONS 9
Starting an operation 9
Getting an operation's status 10
IxLoadGateway - IxLoad session handling 11
Creating a new session 11
Starting a session 12
Deleting a session 15
IxLoad data model 15
Communities 15
Timelines 16

Login name....16
Chassis chain / port assignment operations....16
Adding a chassis....17
Connect to a chassis....18
Removing a chassis....20
Unassign ports....20
Assign ports....20
Statistics....22
Viewing statistics....22
Modifying configured statistics....24
Filter stats....26
Generated CSVs....29
REST Script Templates....29
AddNewCommand....29
ChangeAgentObjectives....30
ChangeIpType....30
SimpleRun....30
IxLoadRestUtils....31
class Connection(__builtin__.object)....31
class WebList(__builtin__.list)....32
class WebObject(__builtin__.object)....32
Functions....33
IxLoadUtils....33
Functions....33

REST resources

A resource is a basic concept in terms of Rest. This chapter will define the resource definition in terms of IxLoad.

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

A resource can have:

- Properties:
  - Primitives: simple types like : bool , int ,string
  - Complex: other resources: timeline resources, agent resources and so on.
- Operations:
  - Example of operations
    - RunTest
    - AddChassis
    - RefreshChassis

Starting with the 8.00 release, IxLoad will have support for REST API. This enables starting and configuring an IxLoad session through REST API, via HTTP requests.

Supported features

Below is a list of features that will be supported in IxLoad 8.00 in REST API mode:

- Create and start an IxLoad session
- Load a configuration (.rxf). The rxf will be loaded from a local path. Uploading an rxf will not be supported in 8.00.
- View data model tree, via GET requests (including query string support)
- Remove existing chassis and add new chassis
- Assign and un-assign ports
- Change existing configuration - modify field values via PATCH requests
- L23 range support - IP ranges
- L47 plugin support - Data protocols
- Save configuration modified through REST API
- View / add / delete configured L47 stats.
- Run test
- Poll L47 statistics

The following list of features will not be supported in IxLoad 8.00 REST API:

- Automatically generated documentation
- Querying logs from REST API
- Uploading RXFs
- Starting remote IxLoad sessions

- Analyzer
- AppLibrary protocols/ Resource Manager / Profiles (e.g. Real Files)
- Other L47 Protocols except Data Protocols
- L23 stacks other than IP
- Adding (POST) / removing (DELETE) objects such as test communities, plugins, ranges. Add and remove operations are only supported officially on the chassis list, the port lists and configured stats.

REST API - Supported methods and running operations

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

REST representation

The Ixload REST api will handle a lot of different object types. Each object will have amongst its values:

- Primitive values, these are basic values;
- Complex values, these will be represented by lists or other REST resources.

Primitive values

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

- strings will be enclosed in quotes : "custom string", ""
- numbers, integer or float will not be enclosed in quotes : 1 , 1.1
- Booleans will not be enclosed in quotes, and will start with lowercase : true, false

List Objects

The IxLoad data model contains numerous lists. In order 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, that 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 8.00. However, this name can be retrieved programatically by performing an 'OPTIONS' request on the resource, and retrieving the value for the 'resourceIdName' field - right now this will return 'objectID'.

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

For a list that has the following URL:

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

An element with objectID = 10 will be retrieved the following URL:

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

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

IxLoad REST API supports the following HTTP methods:

REST resources

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

Case conventions

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

IxLoad REST Methods

IxLoad REST API will support the following HTTP methods - GET, PATCH, POST, DELETE, OPTIONS. The only content type we will support for payloads will be JSON. The Payload applies to PATCH, POST and DELETE methods.

GET

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

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

- 'rel' : the child resource name
- 'href' : a URL at which the child resource can be accessed

Figure 1 shows the output of a GET method applied to the activeTest REST resource in IxLoad.

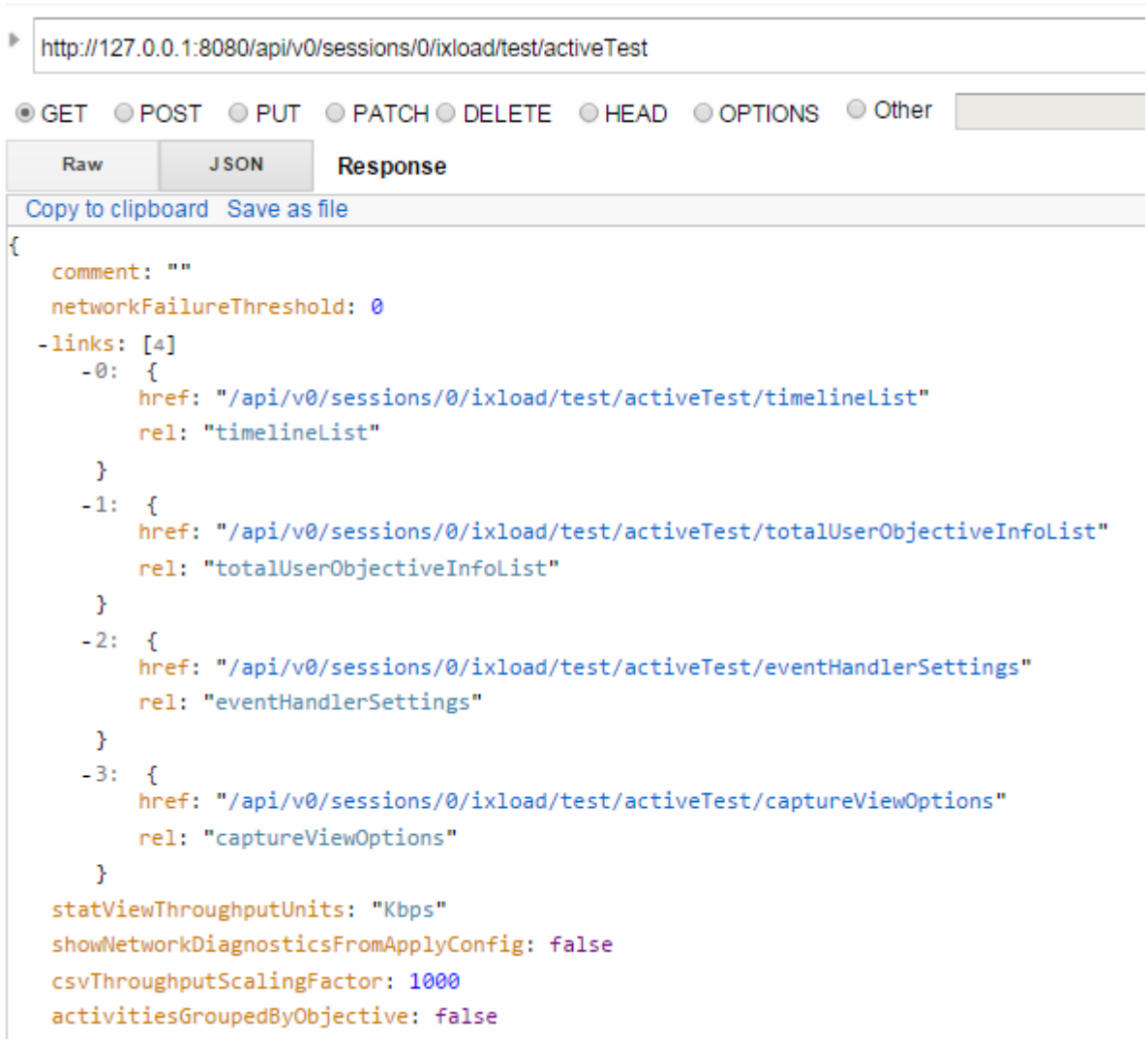

FIGURE 1 GET METHOD

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

PATCH

Users can perform PATCH requests in order to change field values on resources exposed by the IxLoad session. The PATCH request will receive as payload a list of options that the user wishes to modify. Each pair in the dictionary will contain a field name and the new option for it. If the request is successful, a '204 No Content' status will be returned.

The payload for a PATCH request must contain at least one field that will be changed. This means one "field name": "new value" pair. Figure 2 shows the representation of the PATCH method in Advanced REST Client.

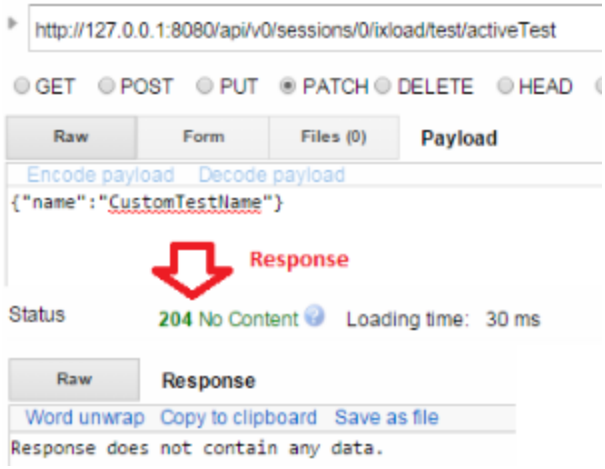

FIGURE 2 PATCH METHOD

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

{

status: "PATCH operation failed"

error: "Cannot change HTTPClient1 at this moment. Please try again later"

}

POST

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

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

Figure 3 Shows the output of the POST method in the Advanced REST Client.

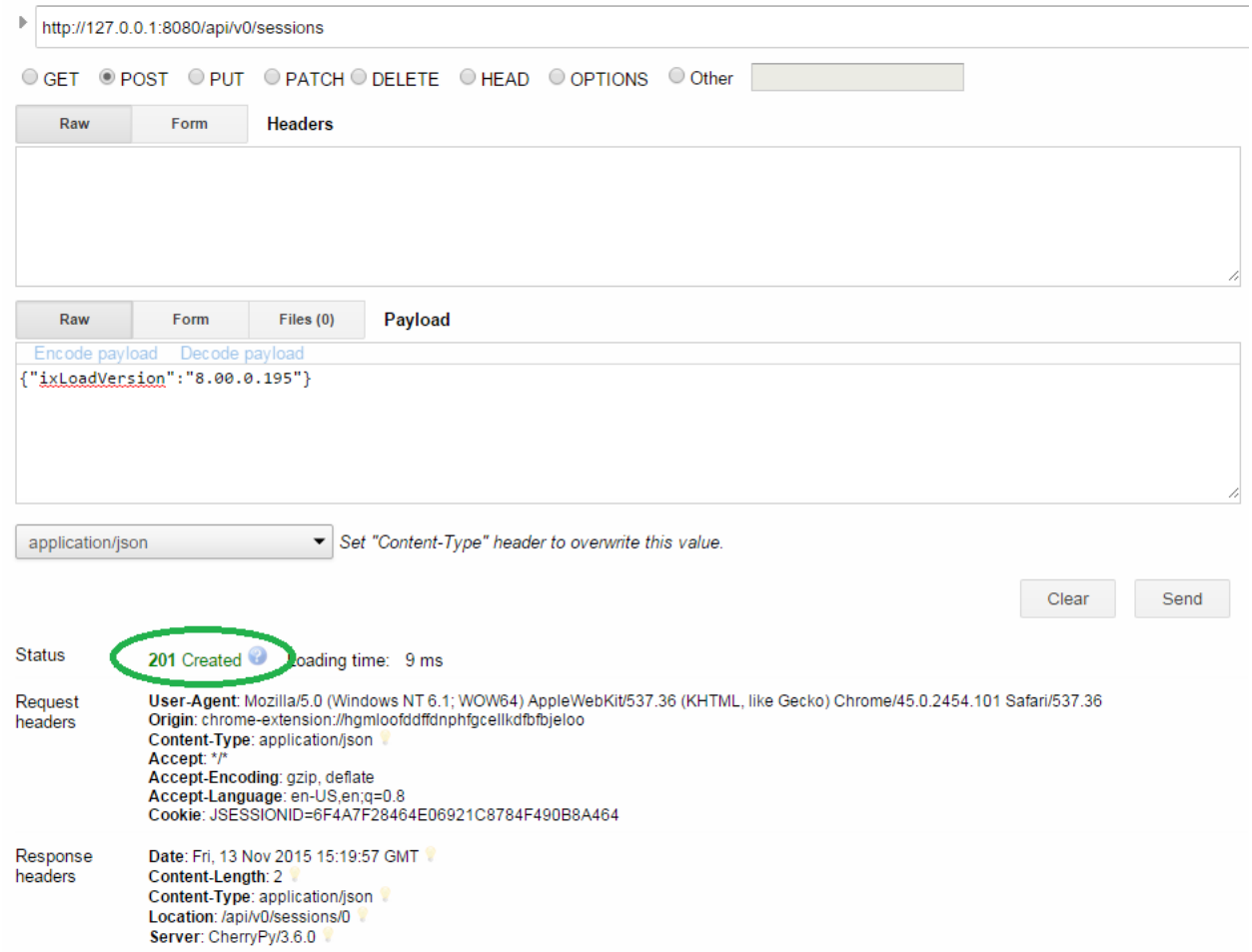

FIGURE 3 POST METHOD

In the response headers there will be a field called 'Location', which will contain 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 / running, a '400 Bad Request' status will be returned.

{

status: "POST operation failed"

error: "Cannot perform the 'POST' operation at this moment. "

}

DELETE

Users can make DELETE requests in order to delete one (or all) of the elements of the list. If the request is successful, a '204 No Content' status will be returned.

DELETE requests don't require any payload.

If the DELETE request is made on a list URL, the list will be cleared - all the elements will be removed.

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

Example 1: DELETE on http://127.0.0.1:8080/api/v0/sessions will delete all sessions

Example 2: DELETE on http://127.0.0.1:8080/api/v0/sessions/2 will only delete 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 / running, a '400 Bad Request' status will be returned.

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

OPTIONS

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

OPTIONS requests don't require any payload.

As previously stated, in the OPTIONS response there will be 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).

Figure 4 shows how the OPTIONS method output looks in the Advanced REST Client.

FIGURE 4 OPTIONS METHOD

OPERATIONS

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

In order to check when operations are available for a certain resource, a GET request can be performed on the resource URL, adding '/operations' at the end of the URL.

Figure 5 shows how the operations available for the test REST resource are represented.

FIGURE 5 OERATIONS GET

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

Starting an operation

Starting an operation is done by performing a POST request on the following URL: $resourceUrl/operations/$operationName. The request payload represents the parameters required by the operation, as shown in the screenshot above. Some operations (such as runTest) may not require any parameters, so for them an empty payload must be sent: {} .

Figure 6 Shows the output of the post command for the loadTest operation.

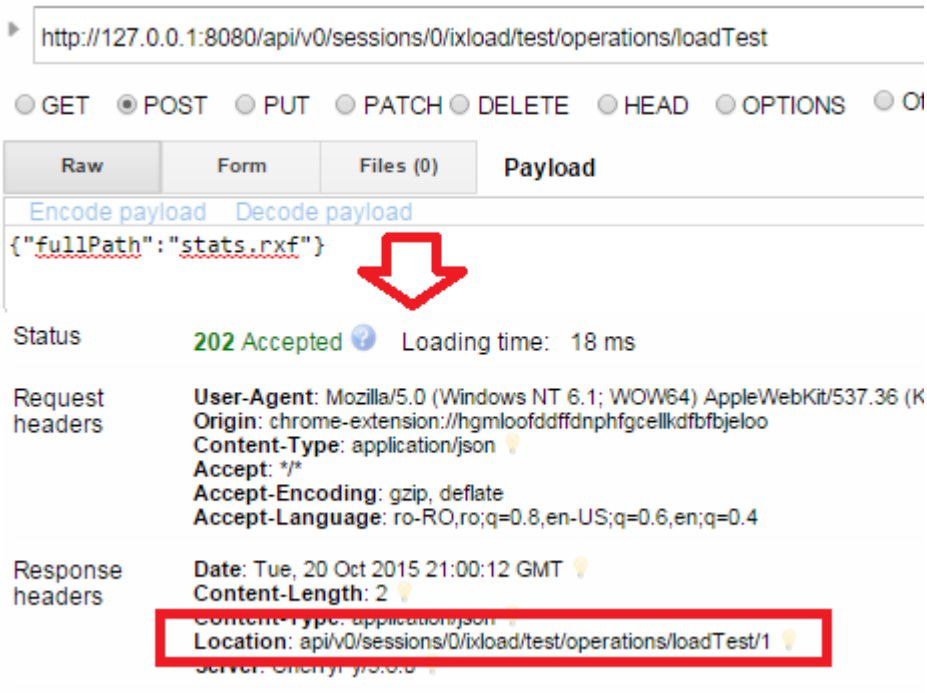

FIGURE 6 OPERATIONS POST

Getting an operation's status

Since these operations are asynchronous methods, users must be able to check an operation's status after they have started it. For this, when starting an operation (executing the POST request), in the response header there will be a field called 'Location', that willl contain a URL. While performing a GET request on that URL, the operation's status will be returned. Figure 7 shows the output for getting the operation status belonging to the loadTest operation.

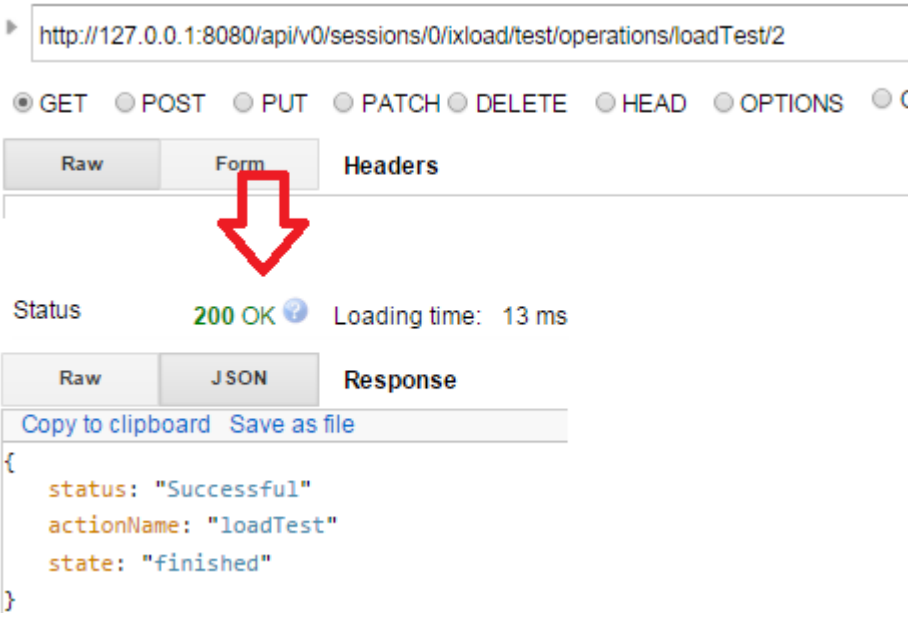

FIGURE 7GET OPERATION STATUS

Possible values for the 'state' field are:

- Created – implies the operation was created;
- Executing – implies the operation is in progress;
- Finished – implies operation is done;

Possible values for the 'status' field are:

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

In case the operation will fail (exit with an error), in the response above a new field will be introduced, that will contain the error message returned by the operation:

{

status: "Error"

actionName: "loadTest"

state: "finished"

error: "File doesn't exist - F:\statsdfs.rxf"

}

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

IxLoadGateway - IxLoad session handling

Creating and handling IxLoad sessions is done through an IxLoad service, named IxLoadGateway.

IxLoadGateway is installed with IxLoad as part of the custom install options.

Creating a new session

In order to create a new session object a POST on api/v0/sessions with payload: {“ixLoadVersion”:”version no.”} Needs to be performed.

Please note that this means the session is just created not started and not active.

This will not take into consideration the instance count limit on the client side. This will only work for sessions that are started. Figure 8 shows how this looks from rest client.

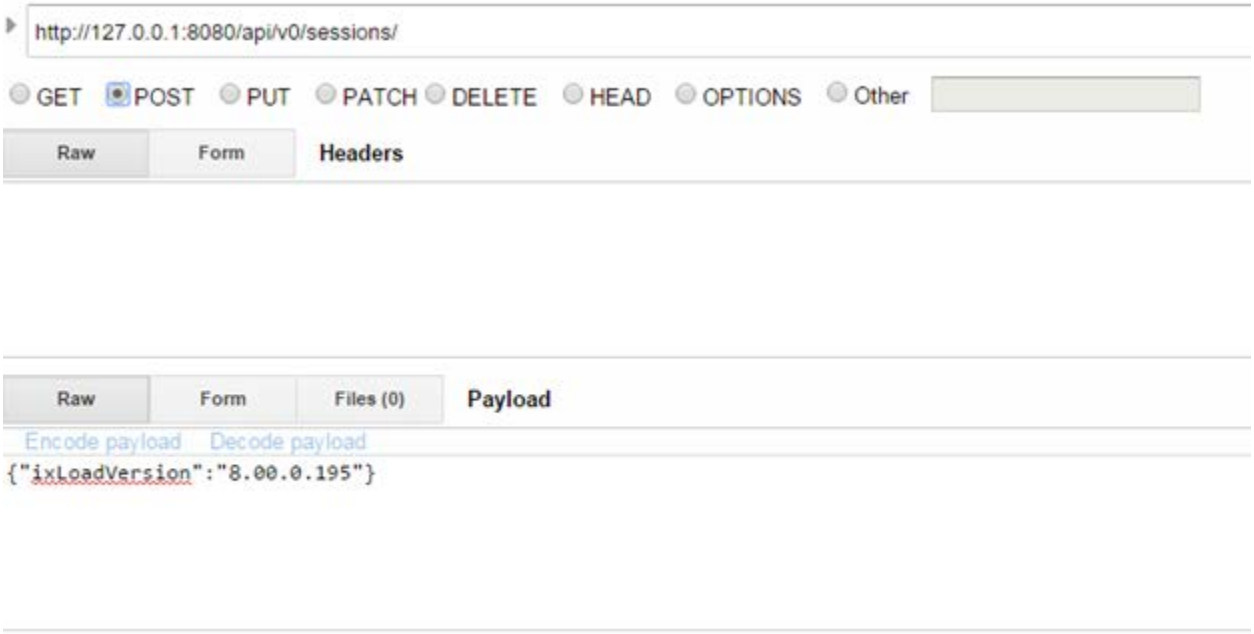

FIGURE 8 CREATE A NEW SESSION OBJECT

Figure 9 shows the response for the post above. Status is 201 created & location points to the session.

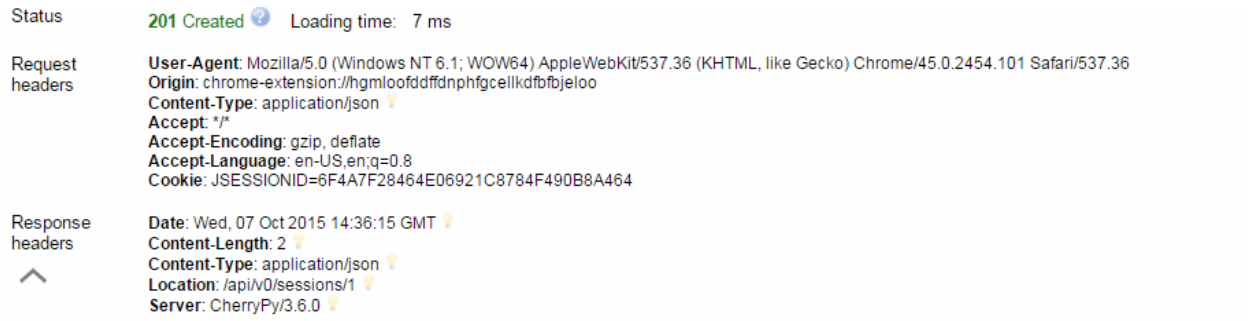

FIGURE 9 RESPONSE FOR CREATING A SESSION OBJECT

Starting a session

To start a session an operation is provided, named *start*. This operation is available on each individual session and requires no payload.

Figure 10 shows how this looks in rest client. This operation will start a new IxLoad session based on the IxLoad version for which the session was created.

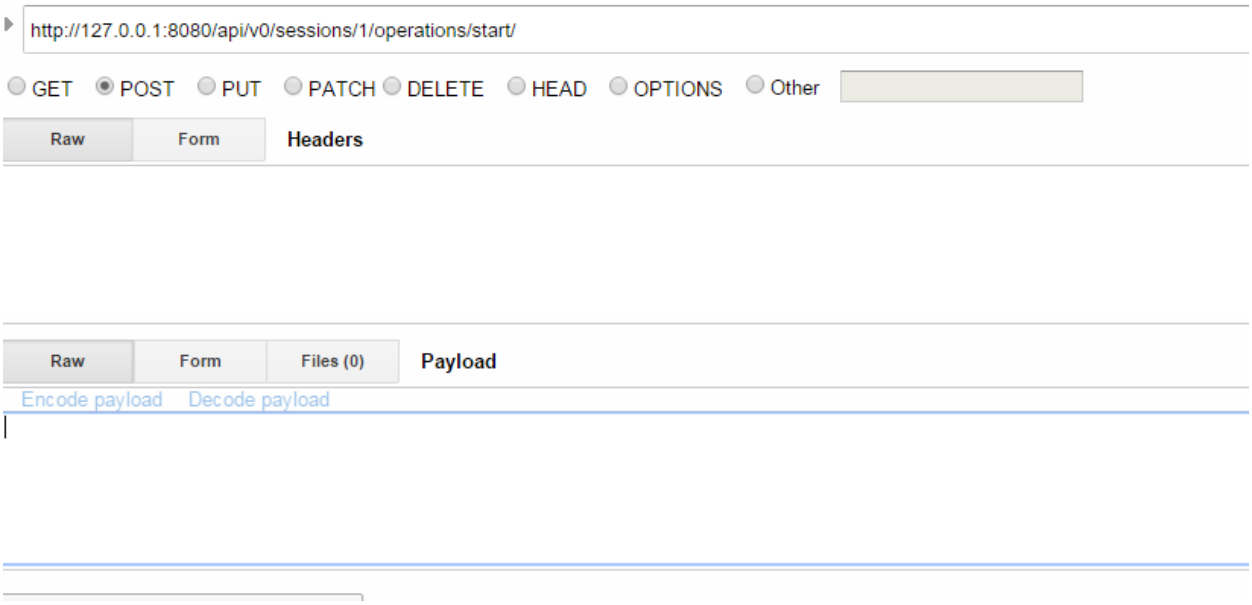

FIGURE 10 START AN EXISTING SESSION

Figure 11 shows the response for the start operation. Response is 202 Accepted. And the location shows the result for the operation.

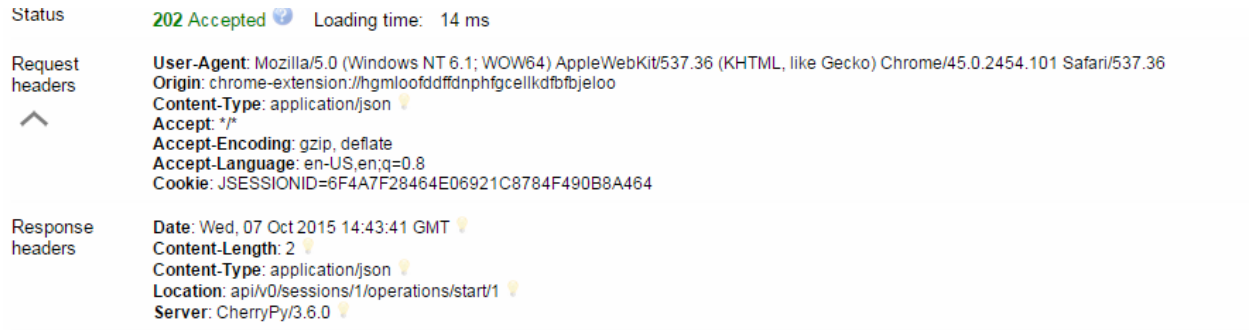

FIGURE 11RESPONSE FOR START OPERATION

Figure 12 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.

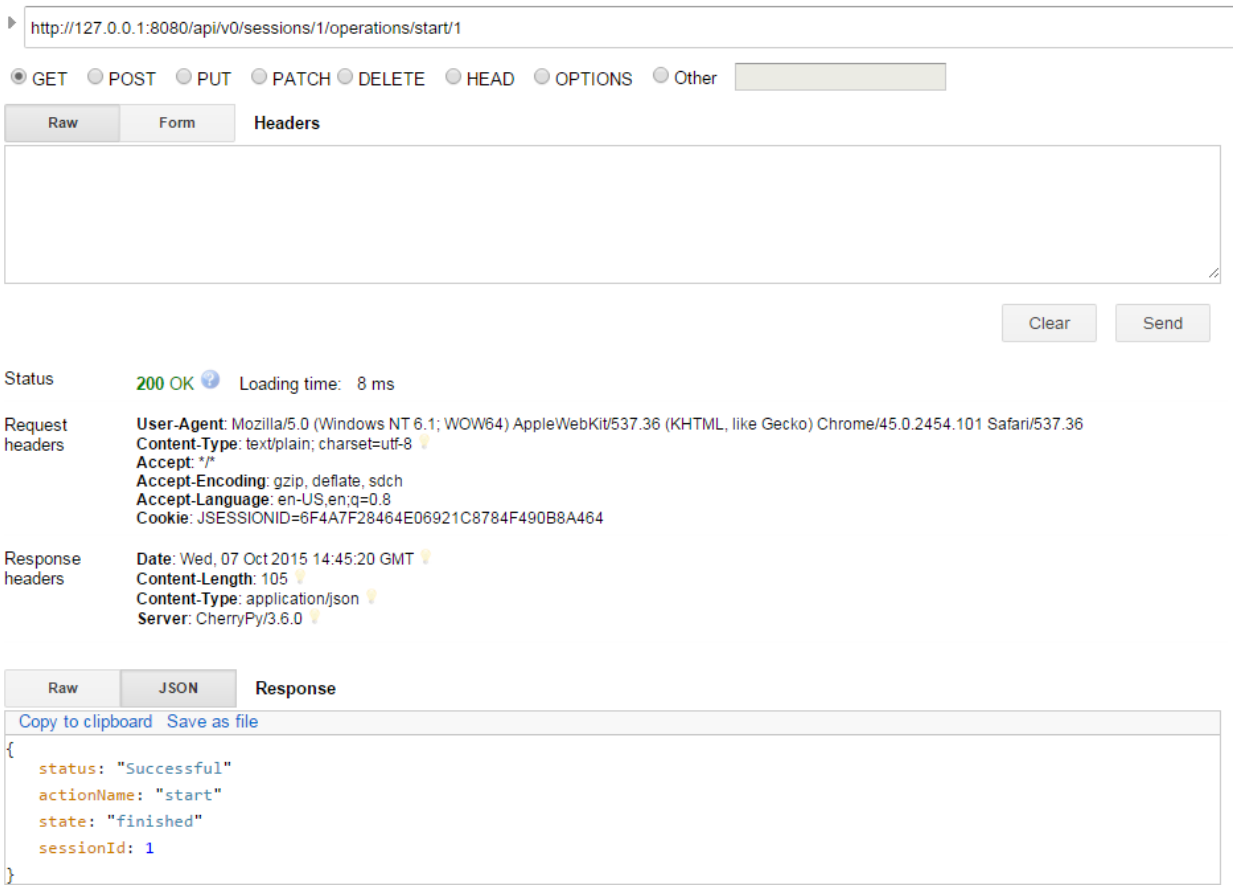

FIGURE 12SUCCESSFUL START

Figure 13 shows the unsuccessful start operation due to maximum number of instances.

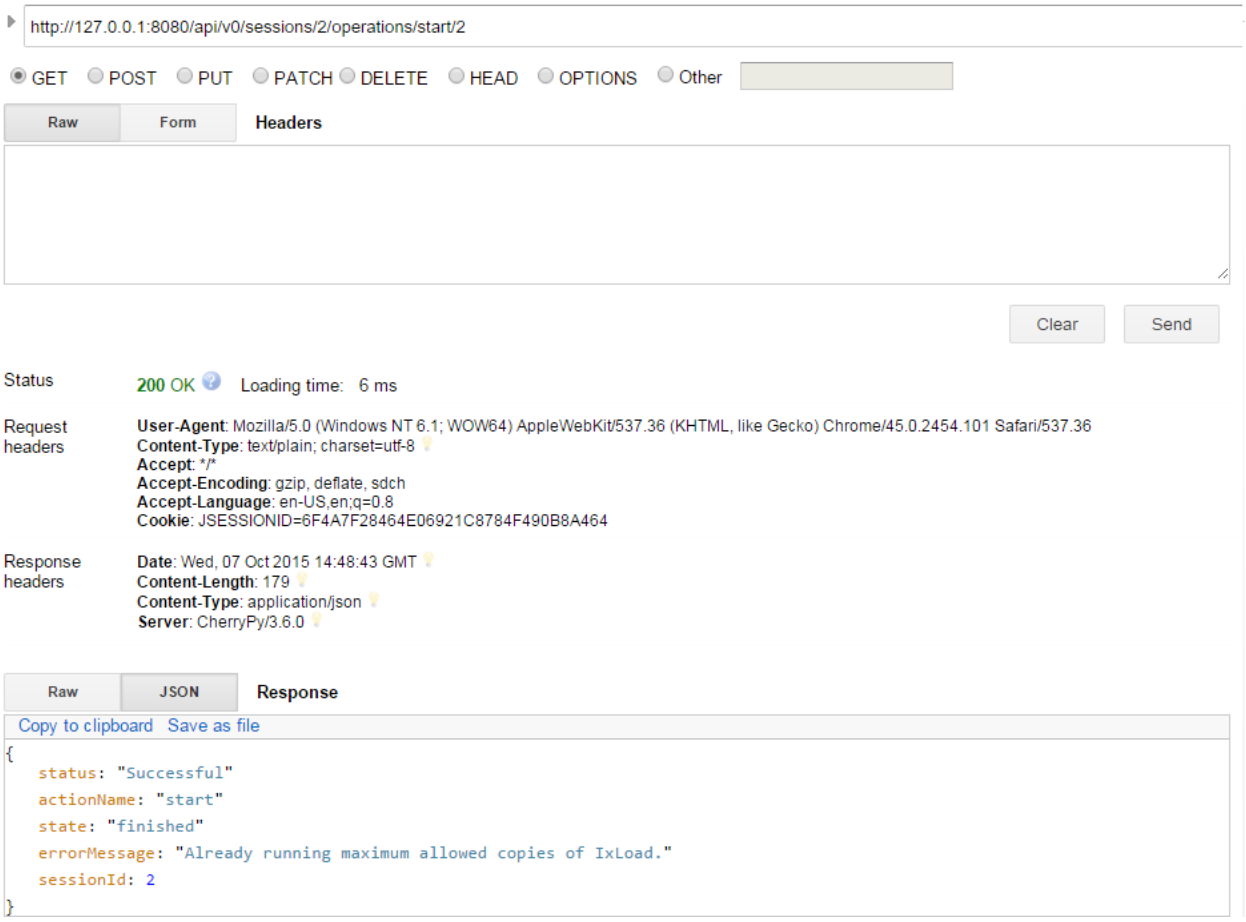

FIGURE 13FAILED SESSION START

Deleting a session

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

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

IxLoad data model

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

Communities

Users can find all the communities in the following path:

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

All the communities in the test will be shown in this list, regardless of their role - client / server / peer. Also, this will contain both enabled and disabled communities.

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

- http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList?role=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 will 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 will be found under the 'network' resource:
  - http://127.0.0.1:8080/api/v0/sessions/0/ixload/test/activeTest/communityList/0/network/stack/childrenList/1/childrenList/1/rangeList
  - '**stack**' is the entry point in the L23 data model.

Timelines

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

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

Plugins will not expose in REST a direct reference to their timeline (the activities don't have a 'timeline' option exposed), they will have a 'timelineId' option. This option will contain the 'objectID' of the desired timeline in the test timeline list. If the user wants to change the timeline used by a certain plugin, he needs to 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

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

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

Chassis chain / port assignment operations

Through IxLoad REST API users can perform the following chassis / port operations:

- Add or remove a chassis
- Connect to a chassis

- Assign or un-assign 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

Adding a new chassis will be done like below:

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

Figure 14 shows the input for rest client.

The newly added chassis is not connected and it has no cards or ports.

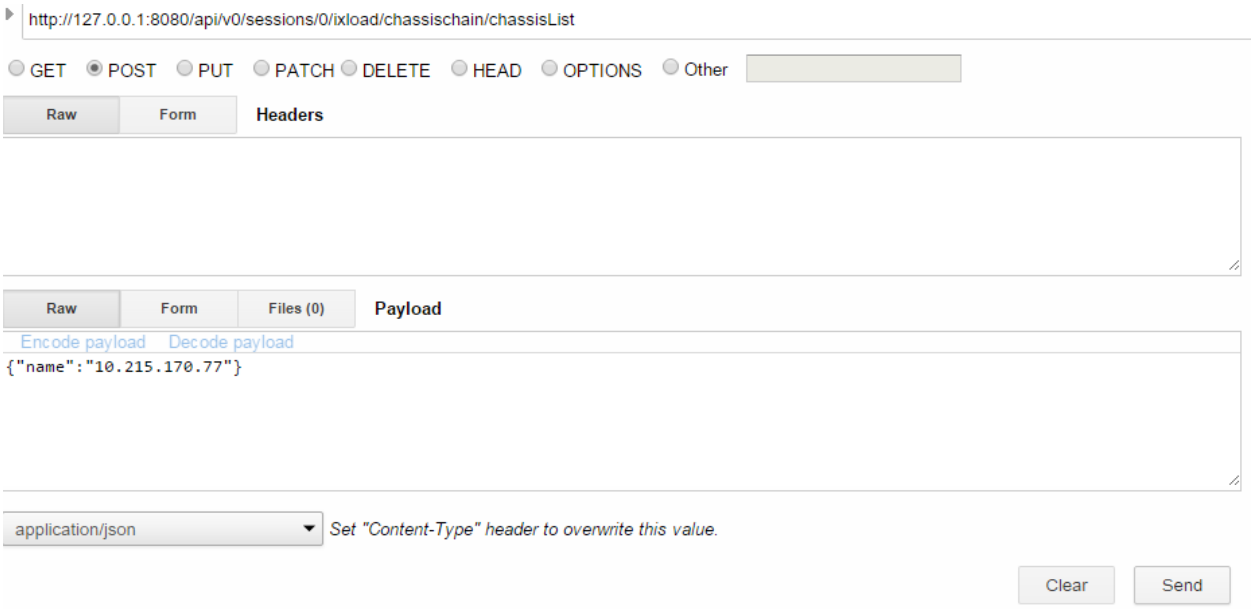

FIGURE 14POST ON CHASSIS LIST

The response for this is shown in Figure 15.

The result is 201 Created.

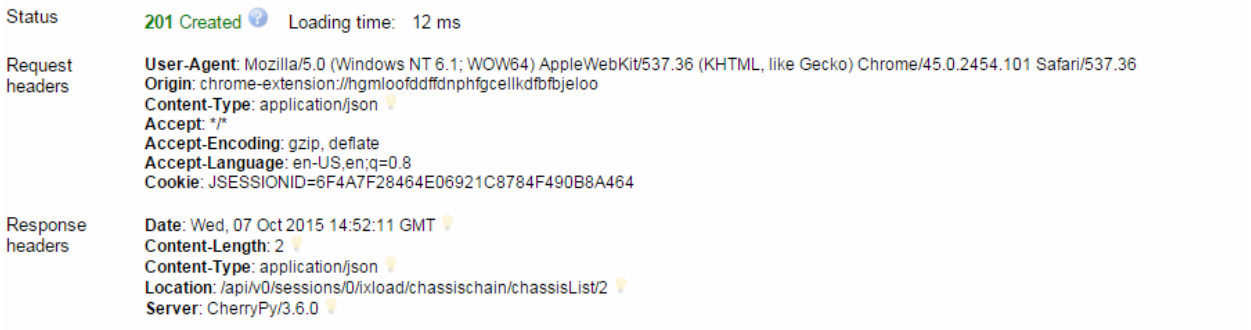

FIGURE 15 RESULT FOR POST ON CHASSISLIST

Connect to a chassis

To connect to a chassis:

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

No payload is required.

Figure 16 shows how this looks in rest client.

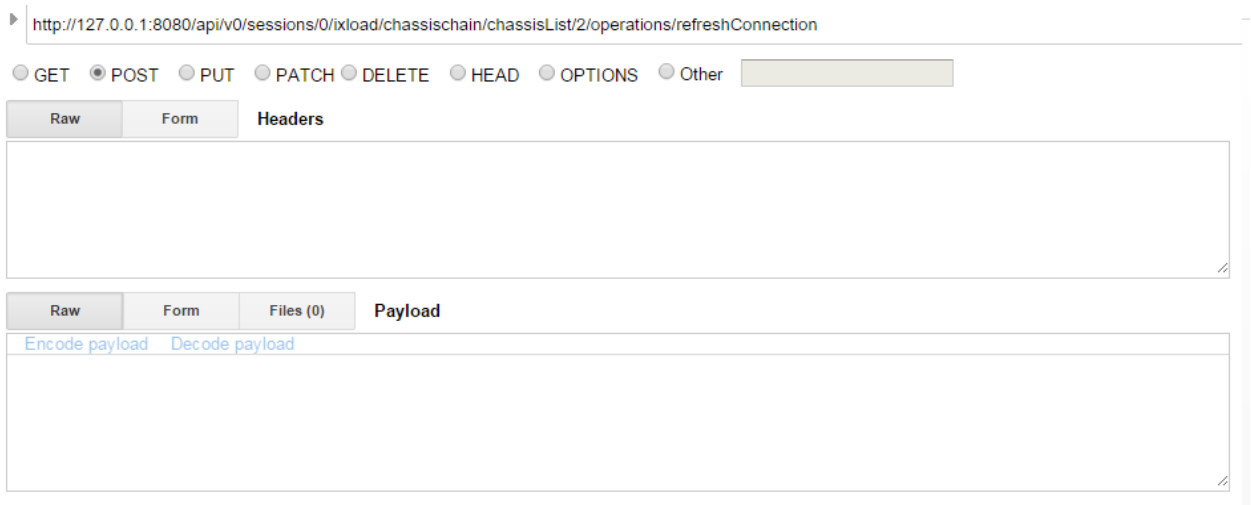

FIGURE 16POST ON REFRESHCONNECTION

Status should be 202 Accepted as in Figure 17.

FIGURE 17 STATUS FOR REFRESHCONNECTION

The result to the refresh operation is posted below in Figure 18.

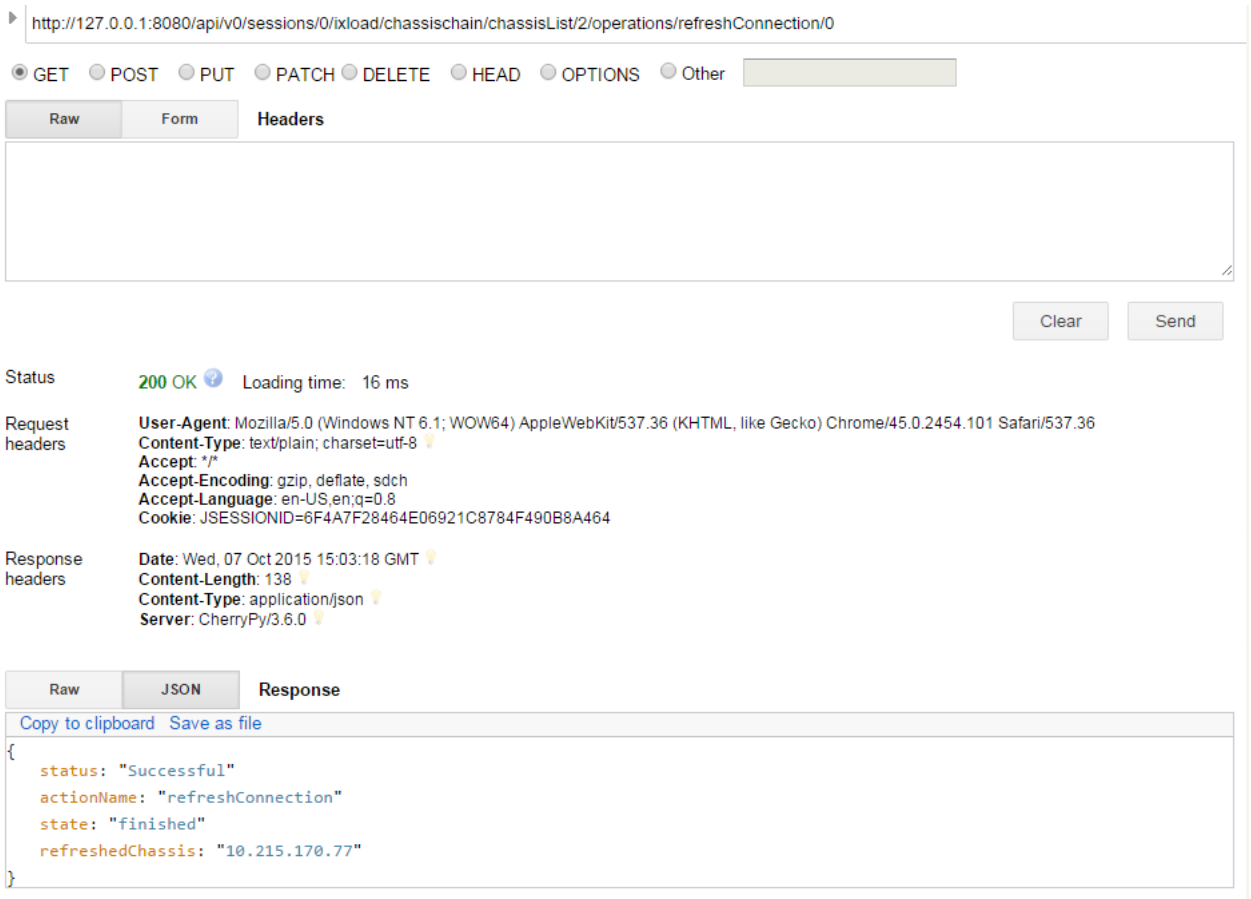

FIGURE 18RESULT TO REFRESHCONNECTION OPERATION

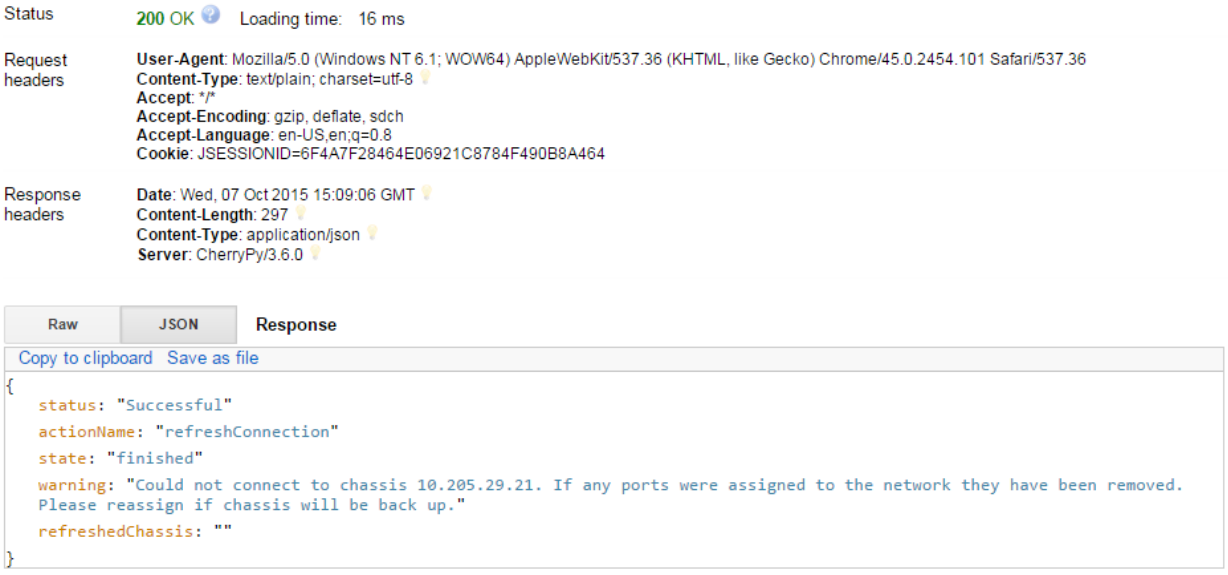

FIGURE 19FAILED TO CONNECT

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

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

There are cases when the rxf will have older chassis that no longer exist. To handle this in the refreshConnection operation a *warning* field was added that will instruct the user on what happened and the *refreshedChassis* field will contain only the chassis that were successfully connected to. In Figure 19 a GET on the status of refreshConnection operation will show that no chassis were refreshed and a warning message telling the user what happened.

Removing a chassis

This is done by performing a simple delete operation on the chassis list. In order to remove all the chassis in the list, the DELETE request must be performed on the chassis list url.

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

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

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

Unassign ports

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

Assign ports

Assigning ports is done by performing POST operations on the network port list. The POST request must receive three parameters: chassisId, cardId, portId. These parameters do **not** represent the unique objectIDs used by REST API to identify resources as part of a list. These three parameters have the same meaning they have from UI and TCL/Python/Perl scripting, where a port is identified using a string such as "1.1.1".
The chassisId, cardId, portId can be seen on performing a GET request on the portList for each card of a chassis as shown in Figure 20.

FIGURE 20PORT LIST

The values highlighted above are the ones that will be used when assigning the port as in Figure 21.

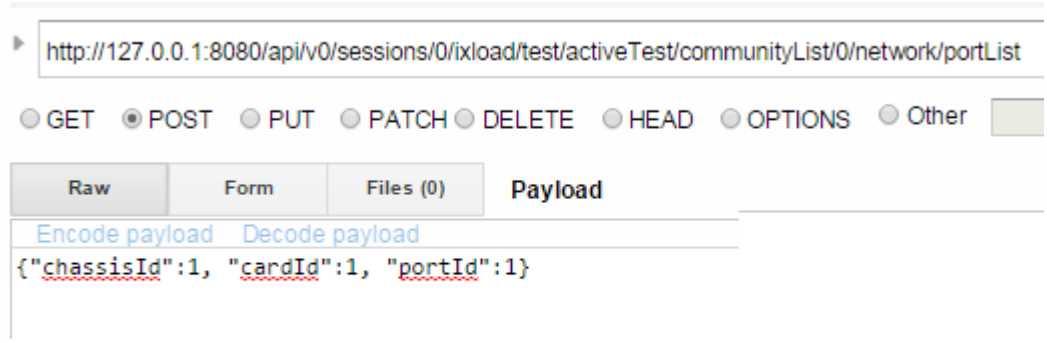

FIGURE 21ASSIGN PORTS

Statistics

The REST stat component will behave similar to the StatCollectorUtils component used in TCL. The user will be able to get available statistics for the activities configured in a test. He will also be able to apply filters on port/nettrafic/activity.

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

Viewing statistics

Using IxLoad REST API users can configure L47 statistics that will be available at run time.
IxLoad REST API offers support for configuring and polling statistics for L47 protocols.

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

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

When a repository is loaded that contains L47 protocols, a GET request on this URL will return a list of stat sources, as seen below in Figure 22.

FIGURE 22STAT SOURCES

A GET request on any of the returned stat sources will return three lists: availableStats, configuredStats and values as below in Figure 23.

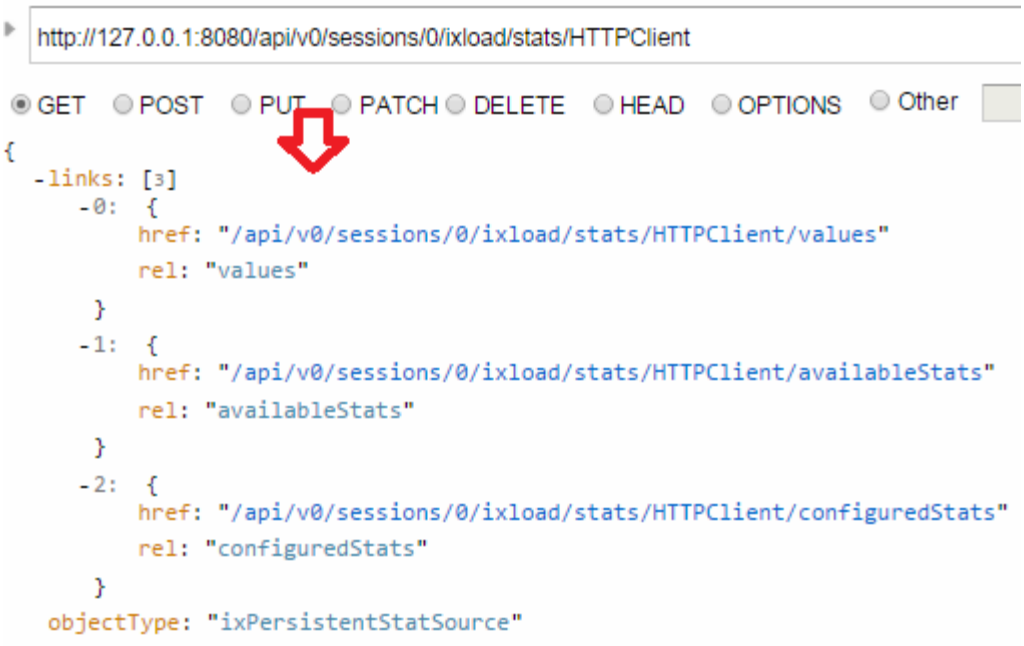

FIGURE 23 GET STAT SOURCE

In Figure 23 there are several complex objects available that refer to how stats work in IxLoad REST framework:

- availableStats: this is the list that contains all available stats for the current test. This list is read-only - users can't remove available stats
- configuredStats: this is the list that contains the stats that have been configured for the current test. Here users can choose to enable / disable / remove / modify existing stats, by default configuredStats will include all availableStats
- Each configured stat resource has the following fields:
  - filterList
  - enabled
  - caption -> (this must be unique in the list)
  - objectID -> (also must be unique)
  - aggregationType
  - statName
- values : this is a dictionary that will contain the actual stat values during the IxLoad test run
  - If a GET request is performed on 'values' before the test actually runs, an empty dictionary will be returned

- The format for the dictionary will be: {timestamp : { stat name : stat value } } The 'values' dictionary will only keep, at all times, the last 20 timestamps. If the user does not poll stats in due time, he might lose some timestamps.

Figure 24 shows the values obtained when running a query on the HTTP Client stat values.

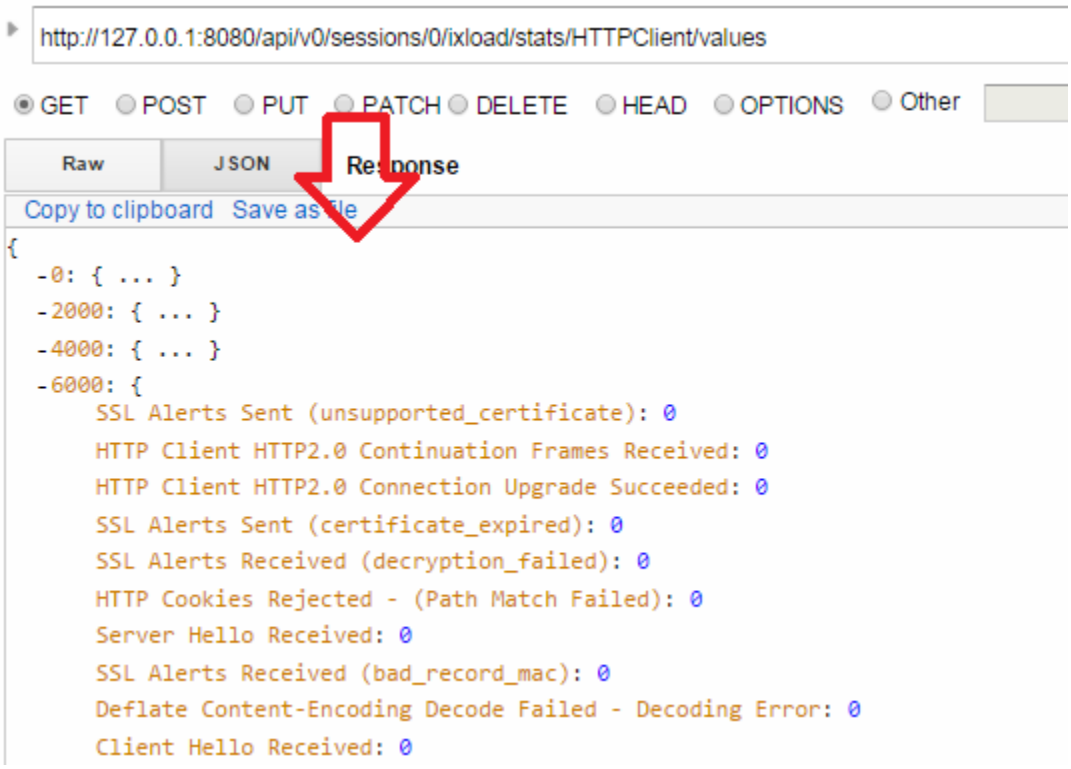

FIGURE 24STAT VALUES

Modifying configured statistics

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

Statistics can be enabled or disabled and the aggregation type can also be changed.

Figure 25 shows the url for getting a configured stat.

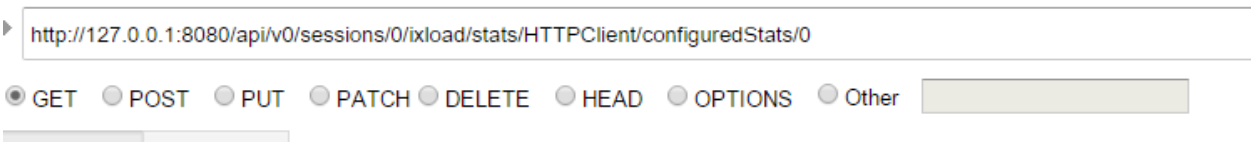

FIGURE 25 GET CONFIGURED STATS

The result of the get in the above request is shown in Figure 26.

Raw JSON **Response**

Copy to clipboard Save as file


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


FIGURE 26 GET CONFIGURED STATS DATA

To change a configured stat a PATCH method will be issued like in Figure 27. The payload must contain the properties that require to be changed.

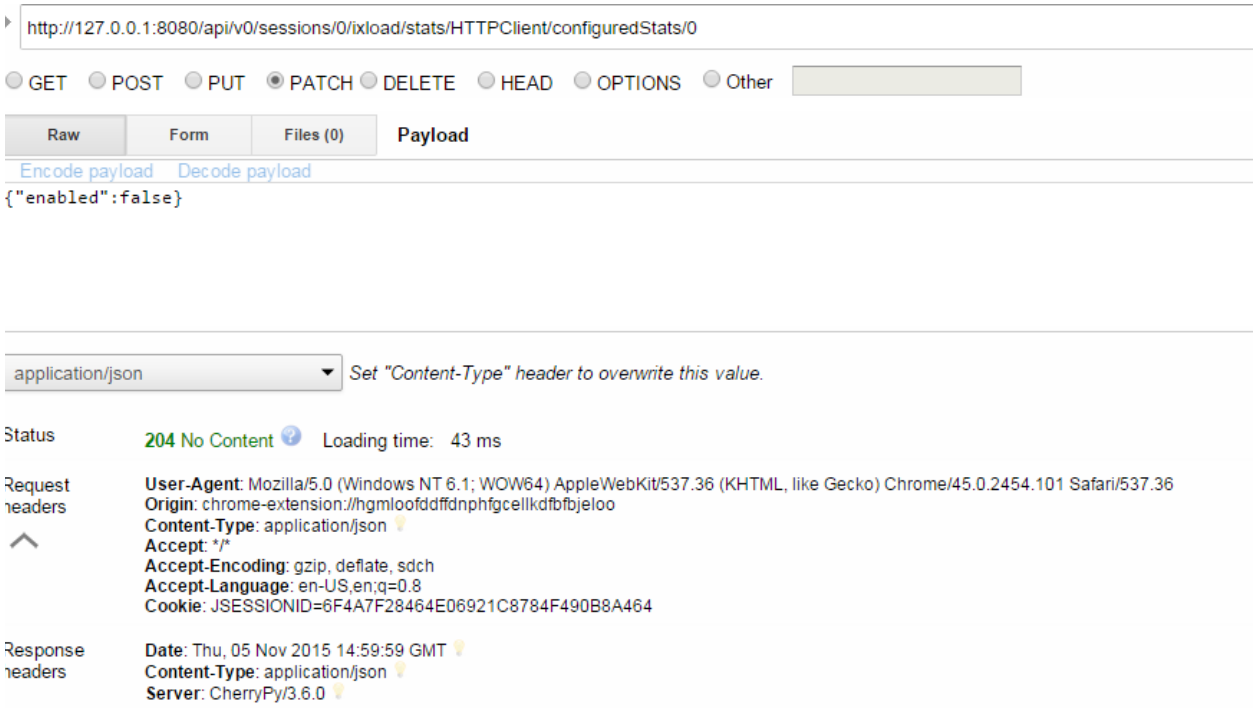

FIGURE 27 PATCH ON CONFIGURED STATS

Figure 28 shows how the PATCH method above changed the configured stat structure by disabling it.

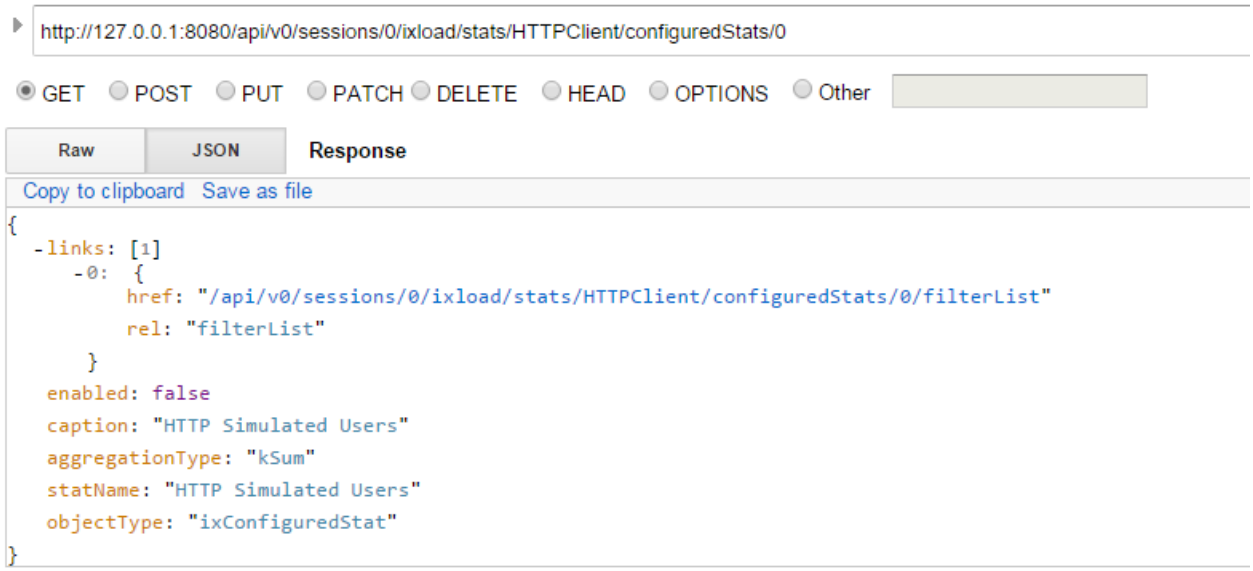

FIGURE 28 CONFIGURED STATS AFTER PATCH

Filter stats

The filter stats are accessed like in **Figure 29** by issuing a GET command on the filterlist from a specific configuredStat item.

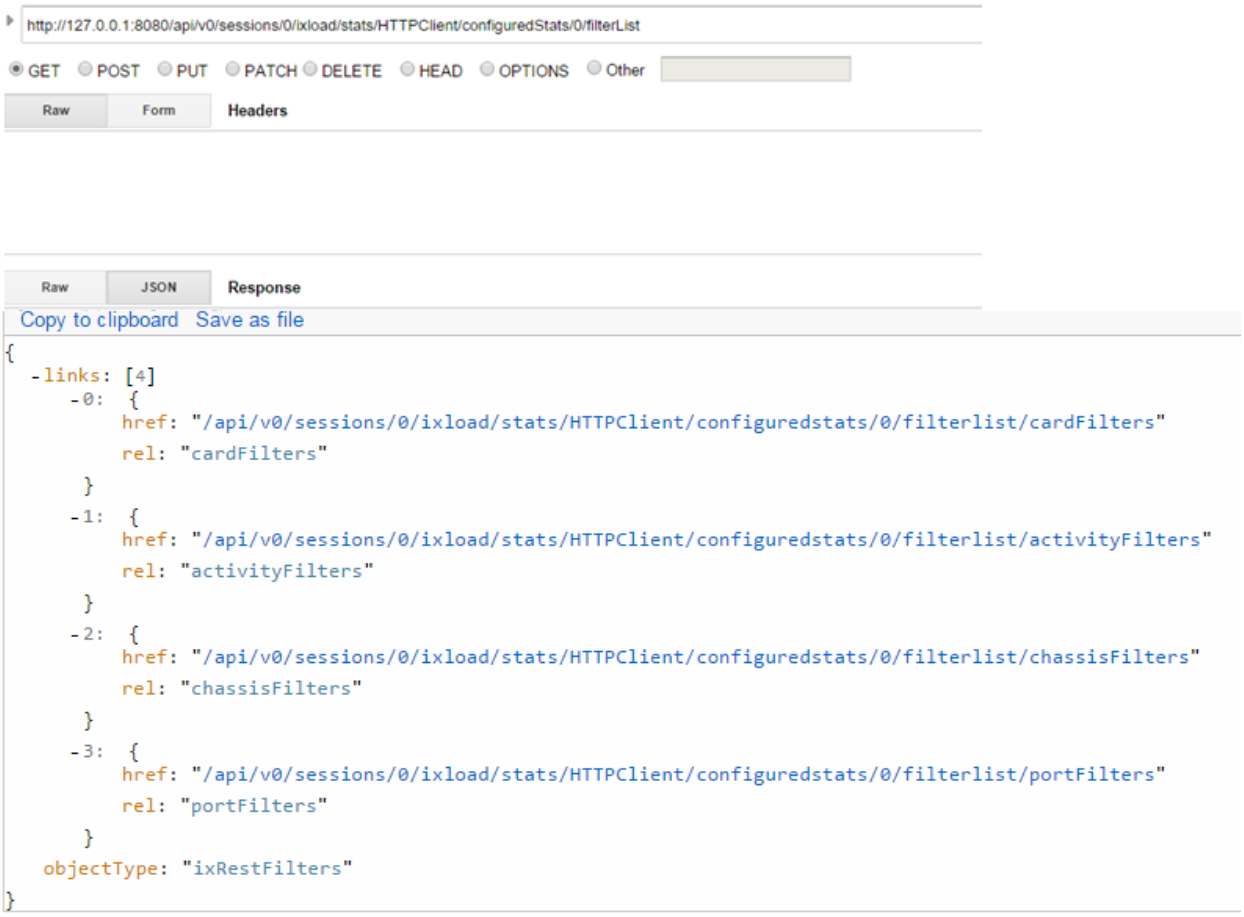

FIGURE 29 CONFIGURED STATS – FILTER LIST

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

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

Adding a port filter is demonstrated in Figure 30 by adding a new port to the portFilter list.

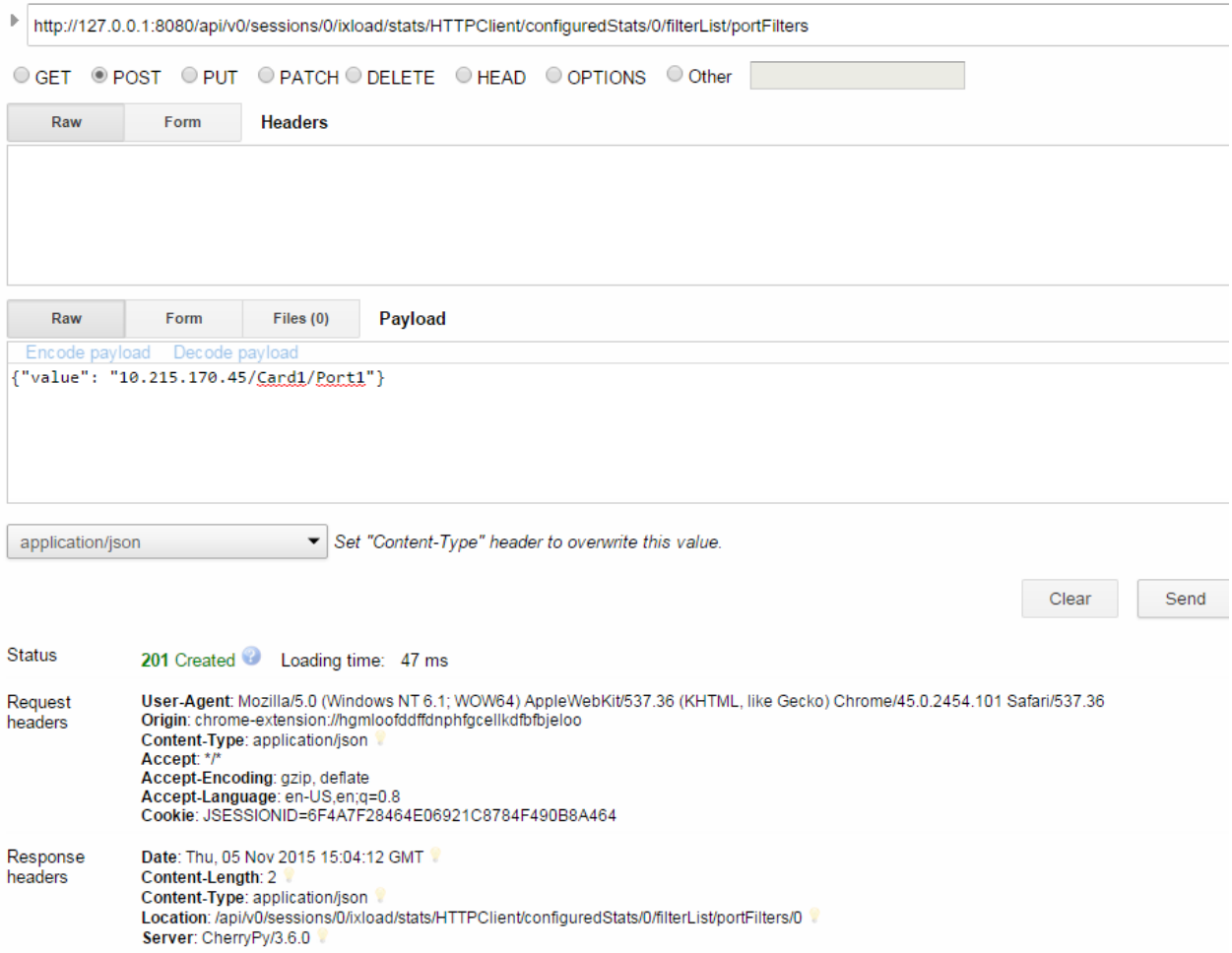

FIGURE 30 POST ON PORT FILTERS

Figure 31 shows how the filter looks after it was added above.

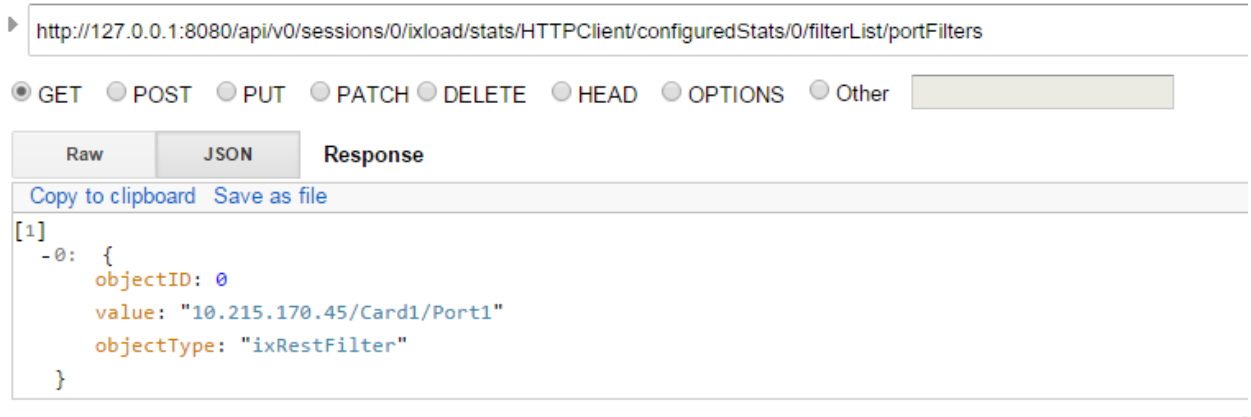

FIGURE 31 GET PORT FILTER

Several filters can beset for multiple configured stats according to how the user will need to see the statistics. Aggregations and processing can be done in the client script once the stats are coming in.

Generated CSVs

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

If the user wants to save the generated CSVs in a custom path, he must set the following two variables on the 'test' resource, before running the configuration:

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

REST Script Templates

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

The sample folder contains 4 python scripts. The 4 scripts located in the samples folder need to be run with a Python executable that has the 'requests' module installed, as it is described in the README.txt file included in the RestScripts folder. The scripts do not require any command line arguments. They can be simply executed by performing'python.exe SimpleRun.py'.

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

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

The REST script templates are relying on 2 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 is using to receive, interpret and dispatch requests.

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

AddNewCommand

This template will:

- Create a session
- Load an Rxf
- Clear the chassis list
- Add a chassis
- Assign ports to the networks
- Clear the command list for client activity

- Update the command list of the client HTTP activity by:
    - Adding a GET command with custom properties
    - Adding a POST command with custom properties
- Save the Rxf
- Start the test
- Poll the stats
- Close the IxLoad session

ChangeAgentObjectives

This template will:

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

ChangeIpType

This template will:

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

SimpleRun

This template will:

- Create a session

- Load an Rxf
- Clear the chassis list
- Add a chassis
- Assign ports to the networks
- Save the Rxf
- Start the test
- Poll the stats
- Close the IxLoad session

IxLoadRestUtils

This module defines the following

class Connection(__builtin__.object)

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

Methods defined here:

- __init__(self, siteUrl, apiVersion)

  Args:

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

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

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

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

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

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

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

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

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

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

  Args:

  - Method (mandatory) represents the HTTP method that will be executed.
  - url (optional) is the url that will be appended to the application url.
  - data (optional) is the data that needs to be sent along with the HTTP method as the JSON payload
  - params (optional) the payload python dict not necessary if data is used.
  - headers (optional) these are the HTTP headers that will be sent along with the request. If left blank will use default

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

Class methods defined here:

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

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

  Currently differs in that it

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

Data descriptors defined here:

- __dict__

  dictionary for instance variables (if defined)

- __weakref__

  list of weak references to the object (if defined)

Data and other attributes defined here:

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

class WebList(__builtin__.list)

By using this class a JSON list will be transformed in a list of WebObject instances.

Methods defined here:

- __init__(self, entries=[])

  Create a WebList from a list of items that are processed by the _WebObject function

Data descriptors defined here:

- __dict__

  dictionary for instance variables (if defined)

- __weakref__

  list of weak references to the object (if defined)

class WebObject(__builtin__.object)

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

Methods defined here:

- __init__(self, **entries)

  Create a WebObject instance by providing a dict having a property - value structure.

- getOptions(self)

  Get the JSON dictionary which represents the WebObject Instance

Data descriptors defined here:

- __dict__

  dictionary for instance variables (if defined)

- __weakref__

  list of weak references to the object (if defined)

Functions

- formatDictToJSONPayload(dictionary)

  Converts a given python dict instance to a string JSON payload that can be sent to a REST API.

- getConnection(server, port)

  Gets a Connection instance, which will be used to make the HTTP requests to the application

IxLoadUtils

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

Functions

- addChassisList(connection, sessionUrl, chassisList)

  This method is used to add one or more chassis to the chassis list.

  Args:

  - connection is the connection object that manages the HTTP data transfers between the client and the REST API
  - sessionUrl is the address of the session that should run the test
  - chassisList is the list of chassis that will be added to the chassis chain.

- addCommands(connection, sessionUrl, commandDict)

  This method is used to add commands to a certain list of provided agents.

Args:

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

- assignPorts(connection, sessionUrl, portListPerCommunity)

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

  Args:

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

- changeActivityOptions(connection, sessionUrl, activityOptionsToChange)

  This method will change certain properties for the provided activities.

  Args:

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

- changeIpRangesParams(connection, sessionUrl, ipOptionsToChangeDict)

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

  Args:

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

- clearAgentsCommandList(connection, sessionUrl, agentNameList)

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

  Args:

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

- clearChassisList(connection, sessionUrl)

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

  Args:

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

- createSession(connection, ixLoadVersion)

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

  Args:

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

- deleteSession(connection, sessionUrl)

  This method is used to delete an existing session.

  Args:

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

- getCommandListUrlForAgentName(connection, sessionUrl, agentName)

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

  Args:

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

- getIPRangeListUrlForNetworkObj(connection, networkUrl)

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

  Args:

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

  - networkUrl is the REST address of the network object for which the network ranges will be provided.

- getTestCurrentState(connection, sessionUrl)

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

  Args:

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

- getTestRunError(connection, sessionUrl)

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

  If no error appeared (the test ran successfully), the return value will be 'None'.

  Args:

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

- loadRepository(connection, sessionUrl, rxfFilePath)

  This method will perform a POST request to load a repository.

  Args:

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

- performGenericDelete(connection, listUrl, payloadDict)

  This will perform a generic DELETE method on a given url

  Args:

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

- performGenericOperation(connection, url, payloadDict)

  This will perform a generic operation on the given url, it will wait for it to finish.

  Args:

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

- performGenericPatch(connection, url, payloadDict)

  This will perform a generic PATCH method on a given url

  Args:

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

- performGenericPost(connection, listUrl, payloadDict)

  This will perform a generic POST method on a given url

  Args:

  - connection is the connection object
  - url is the address of where the operation will be performed
  - payloadDict is the python dict with the parameters for the operation

- pollStats(connection, sessionUrl, watchedStatsDict, pollingInterval=4)

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

  Args:

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

- runTest(connection, sessionUrl)

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

  Args:

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

  - sessionUrl is the address of the session that should run the test.

- saveRxf(connection, sessionUrl, rxfFilePath)

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

  Args:

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

- waitForActionToFinish(connection, replyObj, actionUrl)

  This method waits for an action to finish executing. After a POST request is sent in order to start an action,

  The HTTP reply will contain, in the header, a 'location' field, that contains an URL.

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

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

  Args:

  - connection is the connection object that manages the HTTP data transfers between the client and the REST API
  - replyObj the reply object holding the location
  - actionUrl - the url pointing to the operation

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