# IxChariot API Guide

IxChariot 9.5, August 2017

# Contacting Ixia

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

This page intentionally left blank.

## Contents

[This page intentionally left blank]

# IxChariot WebAPI Guide

The IxChariot WebAPI library is a collection of Python utilities that enable you to create Python scripts that can access the same capabilities as the IxChariot GUI.

Through the WebAPI library, you can create and manage sessions, load and save test configurations, and run tests. The library also provides helpers for accessing web app-specific URLs, and for creating and modifying the JSON (JavaScript Object Notation) data that IxChariot uses.

## Requirements

To use the IxChariot WebAPI, you also need the following software:

- Python, the scripting language and development environment
- virtualenv, a tool for creating isolated ("sandboxed") Python environments, and pip, a tool for installing and managing Python packages (included with virtualenv)
- Requests, an HTTP library for Python
- the WebAPI

You can run the WebAPI scripts from the same PC where IxChariot is installed, or from a separate PC, as long as the separate PC can communicate with the IxChariot host.

| Where you will run WebAPI scripts from | Install: |
|---|---|
| Same PC where IxChariot is installed | • Python 2.7.9 or later version of 2.7 (Python 3 is not supported)<br>• Requests 1.1.0 - 2.3.0 (Requests versions higher than 2.3.0 are not supported)<br>• WebAPI |
| Different PC from where IxChariot is installed | • Unix/Linux, OS X, or Windows (XP or later)<br>• Python 2.7.9 or later version of 2.7 (Python 3 is not supported)<br>• Requests 1.1.0 - 2.3.0 (Requests versions higher than 2.3.0 are not supported)<br>• WebAPI |

## Installation

This section describes how to install the IxChariot WebAPI and its supporting software.

## Installing Python

1. Download Python 2.7.9 or later version of 2.7 from the Python website. WebAPI does not support Python 3.

2. Follow the instructions on the website to install Python.
3. After installation is complete, add the Python directory to the system PATH environment variable. The default directory is `C:\Python27`.

## Installing the virtualenv and Requests

### Windows

The following procedure describes one procedure for installing the Requests libraries. The Requests website describes other procedures.

1. Download virtualenv. virtualenv comes with pip and all its prerequisites.
2. Extract the contents of the downloaded zip file to a temporary folder (for example `C:\temp\virtualenv`). You may also need to extract the tar file under the new directory.
3. Open a console window and change the path to the distribution folder (for example: `C:\temp\virtualenv`).
4. Run the virtualenv install script. Type:
   ```
   python setup.py install
   ```
5. After the install script completes, create a Python sandbox. From the same directory, type:
   ```
   python virtualenv.py c:\webapi
   ```
   The sandbox allows you to install things without affecting your main Python installation or other instances of virtualenv .
6. Activate the sandboxed virtualenv (named `webapi` in this example). Type:
   ```
   c:\webapi\scripts\activate
   ```
   After activation, the command prompt displays `(webapi)` ahead of the path (for example: `(webapi) c:\temp>` ). You will need to enter the `activate` command before running any WebAPI scripts. Once the virtualenv is active, pip is available to install almost any Python package, including Requests.
7. Install Requests. Type:
   ```
   python -m pip install -Iv requests==2.3.0
   ```
   After installation, Requests is available.

NOTE WebAPI does not support a requests version higher than 2.3.0.

### Linux

Linux installation should be possible through your release-specific package manager. If Requests is not directly available through the GUI, you should be able to select the pip module with your Python installation, and then use that to install requests (for example, `pip install requests==2.3.0`).

Otherwise, use the Windows installation procedure as a guide - you will need to perform similar steps, but you may also need to perform additional steps such as using the `sudo` command or logging in as `root`.

## Installing the WebAPI

1. Download the WebAPI distribution file:
   a. In the IxChariot GUI, go to **Help>Python API Library** and click the link.
      The WebAPI .zip file is automatically downloaded.
   b. Unzip the file to a folder of your choice.
2. Add the folder where you unzipped the .zip file to the `PYTHONPATH` environment variable. If you add the variable through the Windows GUI, you will need to launch a new command window so that the variable takes effect. If you are using virtualenv, you will need to activate it as described in Installing the virtualenv and Requests.
3. To confirm that everything is installed correctly, start Python, and then from the Python command prompt, type:

   ```
   import ixia.webapi
   ```

   If the installation is correct, the command will succeed (no message appears).

   You can proceed to running an IxChariot sample script.

## Sample Scripts

Several sample scripts are included with the IxChariot WebAPI. You can use these scripts to familiarize yourself with the WebAPI, and as the basis for your own scripts.

## IxChariot API Sample

To illustrate the general structure of an IxChariot API script, this section provides a commented sample script.

The script sample, `ixchariot_sample.py`, is available in the \samples subdirectory under the directory where the API archive was unzipped.

An `ixchariotApi.py` Python module is provided in the same subdirectory.

It is a wrapper over the WebAPI including the IxChariot specific functions and procedures, which must be used conjointly with the `ixchariot_sample.py` to get a general idea about the IxChariot WebAPI structure.

```
from ixia.webapi import *
import ixchariotApi


webServerAddress = "https://ixchariot-server"
apiVersion = "v1"
username = "N/A"
password = "N/A"
apiKey = "N/A"           # Get the API Key from the web interface, Menu > My Account > Api
Key
```

```
print "Connecting to " + webServerAddress
api = webApi.connect(webServerAddress, apiVersion, None, username, password)
# It is also possible to connect with the API Key instead of username and password,
using:
#api = webApi.connect(webServerAddress, apiVersion, apiKey, None, None)

session = api.createSession("ixchariot")
print "Created session %s" % session.sessionId

print "Starting the session..."
session.startSession()

print "Configuring the test..."

# Configure few test options
testOptions = session.httpGet("config/ixchariot/testOptions")
testOptions.testDuration = 20
testOptions.consoleManagementQoS = ixchariotApi.getQoSTemplateFromResourcesLibrary
(session, "Best Effort")
testOptions.endpointManagementQoS = ixchariotApi.getQoSTemplateFromResourcesLibrary
(session, "Best Effort")
session.httpPut("config/ixchariot/testOptions", data = testOptions)

# Available endpoints used in test (list of 'testIP/mgmtIP' strings)
src_EndpointsList = ["192.168.1.100/192.168.1.101"]
dst_EndpointsList = ["192.168.1.200/192.168.1.201"]

# Create a new ApplicationMix
name = "AppMix 1"
objective = "USERS"
users = 20
direction = "SRC_TO_DEST"
topology = "FULL_MESH"
appmix = ixchariotApi.createApplicationMix(name, objective, users, direction, topo-
logy)
session.httpPost("config/ixchariot/appMixes", data = appmix)

# Configure endpoints for the AppMix

# This demonstrates how to manually assign endpoints to the test configuration using
known IP addresses.
# If you want to assign an endpoint discovered by the Registration Server, use the
ixchariotApi.getEndpointFromResourcesLibrary() function
# to get the data for httpPost
for src_Endpoint in src_EndpointsList:
       ips = src_Endpoint.split('/')
       session.httpPost("config/ixchariot/appMixes/1/network/sourceEndpoints", data = 
ixchariotApi.createEndpoint(ips[0], ips[1]))
for dst_Endpoint in dst_EndpointsList:
       ips = dst_Endpoint.split('/')
       session.httpPost("config/ixchariot/appMixes/1/network/destinationEndpoints", data = 
ixchariotApi.createEndpoint(ips[0], ips[1]))

# Add applications to the AppMix

#                        appName                        appRatio
appList = [
                       ["Facebook",                     10],
                       ["Yahoo Mail",                   40],
                       ["YouTube Enterprise",  50]
                 ]

for i in range(0, len(appList)):
       appData = appList[i]
       appName = appData[0]
       appRatio = appData[1]
       appScript = ixchariotApi.getApplicationScriptFromResourcesLibrary(session, appName)
       app = ixchariotApi.createApp(appScript, appRatio);
       session.httpPost("config/ixchariot/appMixes/1/settings/applications", data = app)

# Create a new FlowGroup
name = "FlowGroup 1"
direction = "SRC_TO_DEST"
topology = "FULL_MESH"
flowgroup = ixchariotApi.createFlowGroup(name, direction, topology)
session.httpPost("config/ixchariot/flowGroups", data = flowgroup)

# Configure endpoints for the FlowGroup

# This demonstrates how to manually assign endpoints to the test configuration using 
known IP addresses.
# If you want to assign an endpoint discovered by the Registration Server, use the 
ixchariotApi.getEndpointFromResourcesLibrary() function
# to get the data for httpPost
for src_Endpoint in src_EndpointsList:
       ips = src_Endpoint.split('/')
       session.httpPost("config/ixchariot/flowGroups/1/network/sourceEndpoints", data = 
ixchariotApi.createEndpoint(ips[0], ips[1]))
for dst_Endpoint in dst_EndpointsList:
       ips = dst_Endpoint.split('/')
```

```
	session.httpPost("config/ixchariot/flowGroups/1/network/destinationEndpoints", data
= ixchariotApi.createEndpoint(ips[0], ips[1]))

# Add flows to the FlowGroup

#                               flowName,                                             users,

flowList = [
                                # Data flows
                                ["TCP Baseline Performance",    1,               "TCP",
                                ["UDP Baseline Performance",    3,               "UDP",

                                # VoIP flows
                                ["G.711a (64 kbps)",                    1,               "RTP"

                                # Video over RTP flows
                                ["RTP HD (10 Mbps)",                    1,               "RTP"

                                # Adaptive video over HTTP flows
                                ["Netflix Video SD",                    1,               "TCP"
                        ]

for i in range (0, len(flowList)):
	flowData = flowList[i]
	flowName = flowData[0]
	users = flowData[1]
	protocol = flowData[2]
	sourceQoSName = flowData[3]
	destinationQoSName = flowData[4]
	flowScript = ixchariotApi.getFlowScriptFromResourcesLibrary(session, flowName)

	# Example for changing the parameter values
	if i == 3:
		ixchariotApi.changeFlowScriptParameterValue(flowScript, "Bit Rate", "9.8 Mbps"

	if i == 4:
		ixchariotApi.changeFlowScriptParameterValue(flowScript, "Segment Duration (s)"
"3")

	sourceQoSTemplate = ixchariotApi.getQoSTemplateFromResourcesLibrary(session,
sourceQoSName)
	destinationQoSTemplate = ixchariotApi.getQoSTemplateFromResourcesLibrary(session,
destinationQoSName)

	flow = ixchariotApi.createFlow(flowScript, users, protocol, sourceQoSTemplate,
destinationQoSTemplate)
       session.httpPost("config/ixchariot/flowGroups/1/settings/flows", data = flow)

# Create a new MulticastGroup
name = "MulticastGroup 1"
mcastgroup = ixchariotApi.createMulticastGroup(name)
session.httpPost("config/ixchariot/multicastGroups", data = mcastgroup)

# Configure endpoints for the MulticastGroup
src_EndpointsList = ["192.168.1.100/192.168.1.101"]
dst_EndpointsList = ["192.168.1.200/192.168.1.201", "192.168.1.210/192.168.1.211"]

# This demonstrates how to manually assign endpoints to the test configuration using
known IP addresses.
# If you want to assign an endpoint discovered by the Registration Server, use the
ixchariotApi.getEndpointFromResourcesLibrary() function
# to get the data for httpPost
for src_Endpoint in src_EndpointsList:
       ips = src_Endpoint.split('/')
       session.httpPost("config/ixchariot/multicastGroups/1/network/sourceEndpoints", data
= ixchariotApi.createEndpoint(ips[0], ips[1]))
for dst_Endpoint in dst_EndpointsList:
       ips = dst_Endpoint.split('/')
       session.httpPost("config/ixchariot/multicastGroups/1/network/destinationEndpoints",
data = ixchariotApi.createEndpoint(ips[0], ips[1]))

# Add multicast flows to the MulticastGroup

#                                      mcastFlowName,                  mcastAddress:Port,

mcastFlowList = [
                                       ["Skype-Video-180p",    "224.1.1.1:5000",        "UDP"
                                       ["RTP HD (10 Mbps)",    "224.1.1.2:6000",        "RTP"
                               ]

for i in range (0, len(mcastFlowList)):
       mcastFlowData = mcastFlowList[i]
       flowName = mcastFlowData[0]
       mcastAddressPort = mcastFlowData[1]
       protocol = mcastFlowData[2]
       sourceQoSName = mcastFlowData[3]
       flowScript = ixchariotApi.getFlowScriptFromResourcesLibrary(session, flowName)
       sourceQoSTemplate = ixchariotApi.getQoSTemplateFromResourcesLibrary(session,
sourceQoSName)
       mcastFlow = ixchariotApi.createMulticastFlow(flowScript, mcastAddressPort,
protocol, sourceQoSTemplate)
       session.httpPost("config/ixchariot/multicastGroups/1/settings/flows", data =
mcastFlow)

# As an alternative to creating the test configuration from scratch, you can use
ixchariotApi.loadConfigFromResourcesLibrary()
# to load an existing configuration into the given session, or ixchari-
otApi.importConfigFromFileToResourcesLibrary() followed
# by ixchariotApi.loadConfigFromResourcesLibrary() to use a test configuration file
from disk.

try:
       print "Starting the test..."
       # This is a blocking function which starts the test and waits for it to end
       result = session.runTest()

       print "The test ended"

       # Save all results to CSV files.
       print "Saving the test results into zipped CSV files...\n"
       filePath = "testResults.zip"
       with open(filePath, "wb+") as statsFile:
               api.getStatsCsvZipToFile(result.testId, statsFile)

       # Get results after test run.
       #
       # NOTE: The functions below can also be used to retrieve statistics while the test
is running:
       #   - start the test with the non-blocking function session.startTest() instead of
the blocking session.runTest()
       #   - run a loop until the session.testIsRunning property becomes false
       #   - inside the loop, use the functions below to retrieve the results collected up
to that point
       #         (use time.sleep() to call them at regular intervals)


       # Get test level results.
       # NOTE: use the statistics names from the CSV report (ixchariot.csv)
       results = ixchariotApi.getTestLevelResults(session, ["Throughput"])

       print "Test Level Results: \n"
       for res in results:
               # Each object in the list of results is of type Statistic (contains the statis
name and a list of StatisticValue objects).
               print res.name
```

```
            for val in res.values:
                    # The list will contain StatisticValue objects for all the reported ti
since the beginning of the test.
                    # Each StatisticValue object contains the timestamp and the actual val
                    print str(val.timestamp) + "      " + str(val.value)
            print ""

        # Get group level results.
        # NOTE: use the statistics names from the CSV report (ixchariot_mix.csv)
        results = ixchariotApi.getGroupLevelResults(session, ["Throughput"], "AppMix 1")

        print "Group Level Results for AppMix 1:\n"
        for res in results:
                # Each object in the list of results has a printing function defined.
                # It will print the name of the statistic and the list of timestamp - value pa
                # For accessing each of these components separately see the example above.
                print res
                print ""

        # Get flow level results
        # NOTE: use the statistics names from the CSV report (ixchariot_mix_applic-
ation.csv)
        results = ixchariotApi.getFlowLevelResults(session, ["Throughput", "Total Datagrams
Sent"], "FlowGroup 1", "RTP HD (10 Mbps)", "RTP")

        print "Flow Level Results for RTP HD (10 Mbps) (RTP) from FlowGroup 1:\n"
        for res in results:
                print res
                print ""

        # Get user level results for the first user
        # NOTE: use the statistics names from the CSV report (ixchariot_mix_application_
user.csv)
        results = ixchariotApi.getUserLevelResultsFromFlow(session, ["Throughput"],
"FlowGroup 1", "RTP HD (10 Mbps)", "RTP", 1)

        print "User Level Results for the first user in flow RTP HD (10 Mbps) (RTP),
FlowGroup 1:\n"
        for res in results:
                print res
                print ""

        # Get User Info results for the first user
        # NOTE: use the statistics names from the CSV report (ixchariot_user_info.csv)
        results = ixchariotApi.getUserInfoFromFlow(session, ["Direction", "Source Endpoint
OS", "Error"], "FlowGroup 1", "RTP HD (10 Mbps)", "RTP", 1)

       print "User Info Results for the first user in flow RTP HD (10 Mbps) (RTP),
FlowGroup 1:\n"
       for res in results:
               print res
               print ""

except Exception, e:
       print "Error", e

print "Stopping the session..."
session.stopSession()

print "Deleting the session..."
session.httpDelete()
```

## ixchariotApi.py

```
from ixia.webapi import *

def getResourceFromLibrary(session, resourceCategory, resourceName):
       resources = session.parentConvention.httpGet("ixchariot/resources/" + resourceCat-
egory)
       for i in range (0, len(resources)):
               resource = resources[i]
               if resourceName == resource.name:
                       return resource
       return None

def getEndpointFromResourcesLibrary(session, endpointName):
       return getResourceFromLibrary(session, "endpoint", endpointName)

def getQoSTemplateFromResourcesLibrary(session, qosTemplateName):
       if qosTemplateName == "None":
               qosTemplate = WebObjectBase()
               qosTemplate.name = "None"
               qosTemplate.serviceType = "BEST_EFFORT"
               qosTemplate.type = "NO_QOS"
       else:
               qosTemplate = getResourceFromLibrary(session, "qostemplate", qosTemplateName)
       return qosTemplate

def getFlowScriptFromResourcesLibrary(session, flowName):
       return getResourceFromLibrary(session, "flowscript", flowName)
```

```
def getApplicationScriptFromResourcesLibrary(session, applicationName):
       return getResourceFromLibrary(session, "applicationscript", applicationName)

def getFlowScriptParameter(flowScript, parameterName):
       for i in range (0, len(flowScript.scriptParameters)):
               parameter = flowScript.scriptParameters[i]
               if parameterName == parameter.caption:
                       return parameter
       return None

def changeFlowScriptParameterValue(flowScript, parameterName, parameterValue):
       parameter = getFlowScriptParameter(flowScript, parameterName)
       parameter.value = parameterValue

def createFlowGroup(name, direction, topology):
       flowGroup = WebObjectBase()
       flowGroup.network = createNetwork(direction, topology)
       flowGroup.settings = createFlowGroupSettings(name)
       return flowGroup

def createMulticastGroup(name):
       flowGroup = WebObjectBase()
       flowGroup.network = createNetwork("SRC_TO_DEST", "MULTICAST")
       flowGroup.settings = createMulticastGroupSettings(name)
       return flowGroup

def createApplicationMix(name, distributionType, noUsers, direction, topology):
       appMix = WebObjectBase()
       appMix.network = createNetwork(direction, topology)
       appMix.settings = createApplicationMixSettings(name, distributionType, noUsers)
       return appMix

def createNetwork(direction, topology):
       network = WebObjectBase()
       network.enabled = True
       network.direction = direction
       network.topology = topology
       return network

def createFlowGroupSettings(name):
       settings = WebObjectBase()
       settings.name = name
       return settings

def createMulticastGroupSettings(name):
       settings = WebObjectBase()
       settings.name = name
       return settings

def createApplicationMixSettings(name, distributionType, users):
       settings = WebObjectBase()
       settings.name = name
       settings.distributionType = distributionType
       settings.numberOfUsers = users
       return settings

def createFlow(script, users, protocol, sourceQoS, destinationQoS):
       flow = WebObjectBase()
       flow.script = script
       flow.numberOfUsers = users
       flow.protocol = protocol
       flow.sourceQoS = sourceQoS
       flow.destinationQoS = destinationQoS
       return flow

def createMulticastFlow(script, multicastAddrPort, protocol, sourceQoS):
       mcastFlow = WebObjectBase()
       mcastFlow.script = script
       mcastFlow.multicastIp = multicastAddrPort
       mcastFlow.protocol = protocol
       mcastFlow.sourceQoS = sourceQoS
       return mcastFlow

def createApp(script, ratio):
       app = WebObjectBase()
       app.script = script
       app.ratio = ratio
       return app

def createEndpoint(testIP, mgmtIP):
       endpoint = WebObjectBase()
       endpoint.ips = []
       endpoint.ips.append(createIP(testIP)) # this example demonstrates just 1 test IP,
but it's also possible to add multiple test IPs to the list
       endpoint.managementIp = createIP(mgmtIP)
       endpoint.name = mgmtIP # it's also possible to pass the endpoint name as a para-
meter to the function
       return endpoint

def getIPType(ip):
       if ip.find(":") != -1:
               return "IPV6"
       else:
               return "IPV4"

def createIP(ip_address):
       ip = WebObjectBase()
       ip.address = ip_address
       ip.type = getIPType(ip_address)
       return ip

def saveConfigToResourcesLibrary(session, configName):
       session.saveConfiguration(configName)

def loadConfigFromResourcesLibrary(session, configName):
       session.loadConfiguration(configName)

def exportConfigFromResourcesLibraryToFile(session, configName, filePath):
       # filePath must have the ".ixcfg" extension
       with open(filePath, "wb+") as exportFile:
               session.exportConfigurationToFile(configName, exportFile)

def importConfigFromFileToResourcesLibrary(session, filePath):
       # filePath must have the ".ixcfg" extension (for configs exported from IxChariot
9.3 or newer) or the ".zip" extension (for configs exported from IxChariot 8.0 -
9.2)
       with open(filePath, "rb") as importFile:
               importedConfig = session.importConfigurationFromFile(importFile)
               importedConfigName = importedConfig.details.name
               return importedConfigName

def deleteOldestTestResults(apiConnection, userName, howManyTestsToDelete):
       testResults = apiConnection.httpGet("results", params = {"start" : 0, "limit" :
howManyTestsToDelete, "sortColumn" : "starttime", "sortOrder" : "ascending", "fil-
ter" : "userid:%s" % userName})
       for testResult in testResults.testRunInformationList:
               apiConnection.httpDelete("results/%d" % testResult.testRunId)


class StatisticValue:
       """Describes a value for a statistic at a moment in time.
    @param timestamp:          the moment in the test when the value was recorded.
    @param value:              the actual value recorded.
    """
       def __init__(self, timestamp, value):
```

```
                self.timestamp = timestamp
                if value == None:
                        self.value = "N/A"
                else:
                        self.value = value

class Statistic:
        """Describes a statistic from the test.
    @param name:       the name of the statistic.
    @param values:         the list of values for that statistic in time.
    """

        def __init__(self, name):
                self.name = name
                self.values = []

        def __str__(self):
                res = self.name + ":\n"
                for val in self.values:
                        res = res + str(val.timestamp) + "    " + str(val.value) + "\n"
                return res

        def add_value(self, value):
                self.values.append(value)

def getRawResults(session, dataObject, stats, filter, filterErrorMessage):
        try:
                statsList = WebListProxy([WebObjectProxy(definition = "{" + dataObject + ":" +
stat + "}") for stat in stats])
                query = WebObjectProxy(stats=statsList, cacheSize=100000, filter=filter)
                channel = session.httpPost("stats/channels", WebObjectProxy(timeToLive=120), h
ers={"Content-Type" : "application/json", "Accept" : "application/json"})
                query = session.httpPost("stats/channels/%d/queries" % channel.id, query, head
{"Content-Type" : "application/json", "Accept" : "application/json"})
                requestResult = session.httpPost("stats/channels/%d/requests" % channel.id, We
jectProxy(count=100000))
                valuesList = requestResult.data.map.__dict__[query.id]
        except:
                raise ValueError(filterErrorMessage)
         finally:
                 try:
                         session.httpDelete("stats/channels")
                 except:
                         pass
        return (valuesList, query)

def getResults(session, dataObject, stats, filter, filterErrorMessage):
        (valuesList, query) = getRawResults(session, dataObject, stats, filter, fil-
terErrorMessage)

        statsResults = []
        for i in range(0, len(stats)):
                result = Statistic(stats[i])
                for j in range(0, len(valuesList)):
                        value = valuesList[j]

                        if len(value.values) == 0:
                                raise ValueError(filterErrorMessage)

                        statValue = StatisticValue(value.timestamp, value.values[0][i])
                        result.add_value(statValue)
                statsResults.append(result)

        return statsResults

def getTestLevelIndexOfFirstUserForGroupApp(session, group, app, fil-
terErrorMessage):
        # Create a filter that will return all the users associated with the given group
and app, for the first timestamp
        filter = WebObjectProxy(
                                type = 'boolean',
                                leftItem = WebObjectProxy(
                                                                leftItem = "ixchariot:mix",
                                                                operator = '=',
                                                                rightItem = group),
                                operator = 'and',
                                rightItem = WebObjectProxy(
                                                                type = 'boolean',
                                                                leftItem = WebObjectProxy(



                                                                operator = 'and',
                                                                rightItem = WebObjectProxy(





        try:
```

```
		(valuesList, query) = getRawResults(session, "ixchariot", ["user"], filter, fi
terErrorMessage)

		# Get the test-level index of the first user for this group and app
		userIndex = min(valuesList[0].values, key=lambda row : int(row[0]))[0]
	except:
		raise ValueError(filterErrorMessage)

	return userIndex

def getTestLevelIndexOfFirstUserForGroupAppFromUserInfo(session, group, app, fil-
terErrorMessage):
	# Create a filter that will return all the users associated with the given group
and app, for the first timestamp
	filter = WebObjectProxy(
				type = 'boolean',
				leftItem = WebObjectProxy(
								leftItem = "ixchariot_user_inf
								operator = '=',
								rightItem = group),
				operator = 'and',
				rightItem = WebObjectProxy(
								leftItem = "ixchariot_user_inf
								operator = '=',
								rightItem = app))

	try:
		(valuesList, query) = getRawResults(session, "ixchariot_user_info", ["User"],
ter, filterErrorMessage)

		# Get the test-level index of the first user for this group and app
		userIndex = min(valuesList[0].values, key=lambda row : int(row[0]))[0]
	except:
		raise ValueError(filterErrorMessage)

	return userIndex

def getTestLevelResults(session, stats):
	"""Gets test level results for the specified statistics.

	Can be used during the test run or after the test has ended.
	Will return all available results, since the beginning of the test.

	@param session:		the current session where test is running / loaded
	@param stats:		the list of statistics names to get the values for (use the
statistics names from the CSV report - ixchariot.csv)

        @return                          a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
                                                Each StatisticValue object contains a statisti
This function will return all the available statistic
                                                values collected since the beginning of the te
Statistic and StatisticValue classes for details.

        @exception ValueError
        """

        filterErrorMessage = "No statistics were reported for this test"
        return getResults(session, "ixchariot", stats, None, filterErrorMessage)

def getGroupLevelResults(session, stats, group):
        """Gets group level results for the specified statistics.

        Can be used during the test run or after the test has ended.
        Will return all available results, since the beginning of the test.

        @param session:         the current session where test is running / loaded
        @param stats:           the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_mix.csv)
        @param group:           the app mix/flow group/multicast group to get the stats for

        @return                          a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
                                                Each StatisticValue object contains a statisti
This function will return all the available statistic
                                                values collected since the beginning of the te
Statistic and StatisticValue classes for details.

        @exception ValueError
        """

        filter = WebObjectProxy(leftItem = "ixchariot:mix", operator = "=", rightItem =
group)
        filterErrorMessage = "Could not find any values for mix/group " + group
        return getResults(session, "ixchariot", stats, filter, filterErrorMessage)

def getAppLevelResults(session, stats, group, app):
        """Gets application level results for the specified statistics.

        Can be used during the test run or after the test has ended.
```

```
	Will return all available results, since the beginning of the test.

	@param session:		the current session where test is running / loaded
	@param stats:		the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_mix_application.csv)
	@param group:		the app mix that contains the app
	@param app:			the app to get the stats for

	@return			a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
					Each StatisticValue object contains a statisti
This function will return all the available statistic
					values collected since the beginning of the te
Statistic and StatisticValue classes for details.

	@exception ValueError
	"""

	filter = WebObjectProxy(
				type = 'boolean',
				leftItem = WebObjectProxy(
								leftItem = "ixchariot:mix",
								operator = '=',
								rightItem = group),
				operator = 'and',
				rightItem = WebObjectProxy(
								leftItem = "ixchariot:applicat
								operator = '=',
								rightItem = app))
	filterErrorMessage = "Could not find any values for mix/group " + group + " and
app/flow " + app
	return getResults(session, "ixchariot", stats, filter, filterErrorMessage)

def getFlowLevelResults(session, stats, group, flow, protocol):
	"""Gets flow level results for the specified statistics.

	Can be used during the test run or after the test has ended.
	Will return all available results, since the beginning of the test.

	@param session:		the current session where test is running / loaded
	@param stats:		the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_mix_application.csv)
	@param group:		the flow group / multicast group that contains the flow
	@param flow:		the flow to get the stats for
	@param protocol:	the test protocol for the flow
	@return				a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
						Each StatisticValue object contains a statisti
This function will return all the available statistic
						values collected since the beginning of the te
Statistic and StatisticValue classes for details.

	@exception ValueError
	"""

	fullFlowName = flow + " (" + protocol + ")"
	return getAppLevelResults(session, stats, group, fullFlowName)

def getUserLevelResultsFromApp(session, stats, group, app, user):
	"""Gets user level results for the specified statistics.

	Can be used during the test run or after the test has ended.
	Will return all available results, since the beginning of the test.

	@param session:		the current session where test is running / loaded
	@param stats:		the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_mix_application_user.csv)
	@param group:		the app mix that contains the app
	@param app:			the app to get the stats for
	@param user:		which user from the app to get results for (a number between 1
the number of users in the app)

	@return				a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
						Each StatisticValue object contains a statisti
This function will return all the available statistic
						values collected since the beginning of the te
Statistic and StatisticValue classes for details.

	@exception ValueError
	"""

	filterErrorMessage = "Could not find any values for mix/group " + group + ", app/-
flow " + app + " and user " + str(user)

	# The user index we receive as input parameter is relative to this group and app
	# We need to convert it to the test-level index
	# (the user index as seen in UI or CSV, taking into account the users from the oth-
ers groups and apps)
```

```
	testLevelUserIndex = int(getTestLevelIndexOfFirstUserForGroupApp(session, group, 
app, filterErrorMessage)) + user - 1

	filter = WebObjectProxy(
				type = 'boolean',
				leftItem = WebObjectProxy(
								leftItem = "ixchariot:mix",
								operator = '=',
								rightItem = group),
				operator = 'and',
				rightItem = WebObjectProxy(
								type = 'boolean',
								leftItem = WebObjectProxy(



								operator = 'and',
								rightItem = WebObjectProxy(



	return getResults(session, "ixchariot", stats, filter, filterErrorMessage)

def getUserLevelResultsFromFlow(session, stats, group, flow, protocol, user):
	"""Gets user level results for the specified statistics.

	Can be used during the test run or after the test has ended.
	Will return all available results, since the beginning of the test.

	@param session:		the current session where test is running / loaded
	@param stats:		the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_mix_application_user.csv)
	@param group:		the flow group / multicast group that contains the flow
	@param flow:		the flow to get the stats for
	@param protocol:	the test protocol for the flow
	@param user:		which user from the flow to get results for (a number between 
the number of users in the flow)

	@return				a list of Statistic objects (one object for each reque
Each Statistic object contains a list of StatisticValue objects.
						Each StatisticValue object contains a statisti
This function will return all the available statistic 
						values collected since the beginning of the te
Statistic and StatisticValue classes for details.
```

```
        @exception ValueError
        """

        fullFlowName = flow + " (" + protocol + ")"
        return getUserLevelResultsFromApp(session, stats, group, fullFlowName, user)

def getUserInfoFromApp(session, stats, group, app, user):
        """Gets User Info for the specified statistics (e.g. Name, Location, Version, Oper-
ating System, IP addresses and errors for the source and destination endpoints).

        Can be used during the test run or after the test has ended.

        @param session:         the current session where test is running / loaded
        @param stats:           the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_user_info.csv)
        @param group:           the app mix that contains the app
        @param app:                     the app to get the stats for
        @param user:            which user from the app to get results for (a number between 1
the number of users in the app)

        @return                         a list of Statistic objects (one object for each reque
Each Statistic object contains a list with one StatisticValue object.
                                                The StatisticValue object contains a statistic
definition of the Statistic and StatisticValue classes for details.
                                                Unlike the other statistics, the User Info res
timestamp. Timestamp 0 is arbitrarily used to keep the same API
                                                as for the other statistics. During the test e
might be added or updated (by overwriting the previous values).

        @exception ValueError
        """

        filterErrorMessage = "Could not find any values for mix/group " + group + ", app/-
flow " + app + " and user " + str(user)

        # The user index we receive as input parameter is relative to this group and app
        # We need to convert it to the test-level index
        # (the user index as seen in UI or CSV, taking into account the users from the oth-
ers groups and apps)
        testLevelUserIndex = int(getTestLevelIndexOfFirstUserForGroupAppFromUserInfo(ses-
sion, group, app, filterErrorMessage)) + user - 1

        filter = WebObjectProxy(
                                type = 'boolean',
                                leftItem = WebObjectProxy(
                                                          leftItem = "ixchariot_user_inf
                                                          operator = '=',
                                                          rightItem = group),
                              operator = 'and',
                              rightItem = WebObjectProxy(
                                                          type = 'boolean',
                                                          leftItem = WebObjectProxy(



                                                          operator = 'and',
                                                          rightItem = WebObjectProxy(



       return getResults(session, "ixchariot_user_info", stats, filter, fil-
terErrorMessage)

def getUserInfoFromFlow(session, stats, group, flow, protocol, user):
       """Gets User Info for the specified statistics (e.g. Name, Location, Version, Oper-
ating System, IP addresses and errors for the source and destination endpoints).

       Can be used during the test run or after the test has ended.

       @param session:         the current session where test is running / loaded
       @param stats:           the list of statistics names to get the values for (use the st
istics names from the CSV report - ixchariot_user_info.csv)
       @param group:           the flow group / multicast group that contains the flow
       @param flow:            the flow to get the stats for
       @param protocol:        the test protocol for the flow
       @param user:            which user from the flow to get results for (a number between
the number of users in the flow)

       @return                         a list of Statistic objects (one object for each reque
Each Statistic object contains a list with one StatisticValue object.
                                       The StatisticValue object contains a statistic
definition of the Statistic and StatisticValue classes for details.
                                       Unlike the other statistics, the User Info res
timestamp. Timestamp 0 is arbitrarily used to keep the same API
                                       as for the other statistics. During the test e
might be added or updated (by overwriting the previous values).

       @exception ValueError
       """
```

```
        fullFlowName = flow + " (" + protocol + ")"
        return getUserInfoFromApp(session, stats, group, fullFlowName, user)
```

## Accessing the API

The IxChariot WebAPI library is exposed through the `ixia.webapi` module. To use it in a script, you import it using the name `ixia.webapi`.

For example:

```
   import ixia.webapi as webapi
```

## Connection Methods

| To... | Use... |
|---|---|
| Create a session | createSession |
| Get the list of saved test configurations | getConfigurations |
| Get the list of sessionIds for current sessions | getSessions |
| Get the list of allowed session types | getSessionTypes |
| Retrieve a set of statistics and store them in a ZIP file | getStatsCsvZipToFile |
| Return an existing session | joinSession |

### createSession

Creates and returns a session of the given type. On creation, sessions are put into the "Initial" state. After being created, sessions must be started.

#### Return Value

Session

#### Syntax

```
createSession(sessionType: string)
```

#### Parameters

| | |
|---|---|
| `sessionType` | Type of session to create:<br>`ixchariot`: IxChariot application |

## getConfigurations

Returns a list of objects that have name and description properties for all saved configurations.

### Return Value

Configuration list

### Syntax

```
getConfigurations()
```

### Parameters

None.

## getSessions

Returns a list of session IDs for sessions that currently exist.

### Return Value

SessionData list

### Syntax

```
getSessions()
```

### Parameters

None

## getSessionTypes

Returns a list of strings allowed for the `sessionType` field of createSession.

### Return Value

String list

### Syntax

```
getSessionTypes()
```

### Parameters

None

## getStatsCsvZipToFile

Retrieves a test's entire set of statistics from the web server and writes them to the file-like object `statFile`.

### Return Value

None

### Syntax

```
getStatsCsvToFile(testOrResultId : int, statFile : fileHandle)
```

### Parameters

| | |
|---|---|
| `testOrResultId` | ID of the test to retrieve statistics for.<br>For example, the ID is returned as the `id` property of the return value from `runTest`. |
| `statFile` | File handle or file-like object to write the CSV-formatted statistics data files to, in ZIP-archive format |

## joinSession

Returns an existing session.

### Return Value

Session

### Syntax

```
joinSession(sessionId: int)
```

### Parameters

| | |
|---|---|
| sessionId | Session to join. |

## IxChariotApi Methods

| To... | Use... |
|---|---|
| Modify the value of a flow parameter | changeFlowScriptParameterValue |
| Create an application | createApp |
| Create an application mix | createApplicationMix |

| To... | Use... |
|---|---|
| Create an endpoint | createEndpoint |
| Create a flow | createFlow |
| Create a flow group | createFlowGroup |
| Create a multicast flow | createMulticastFlow |
| Create a multicast group | createMulticastGroup |
| Delete test results from previous runs | deleteOldestTestResults |
| Save a test configuration from the resources library to a file | exportConfigFromResourcesLibraryToFile |
| Retrieve application level results | getAppLevelResults |
| Get an application from the resources library | getApplicationScriptFromResourcesLibrary |
| Get the list of endpoints from the resources library (including the ones discovered by the Registration Server) | getEndpointFromResourcesLibrary |
| Retrieve flow/multicast flow level results | getFlowLevelResults |
| Get a flow from the resources library | getFlowScriptFromResourcesLibrary |
| Retrieve group/mix level results | getGroupLevelResults |
| Get the list of QoS templates from the resources library | getQoSTemplateFromResourcesLibrary |
| Retrieve test level results | getTestLevelResults |
| Retrieve user level results from an application | getUserLevelResultsFromApp |
| Retrieve user level results from a flow | getUserLevelResultsFromFlow |
| Get user info for the specified statistics from an application | getUserInfoFromApp |
| Get user info for the specified statistics from a flow | getUserInfoFromFlow |
| Load a test configuration from a file to the resources library | importConfigFromFileToResourcesLibrary |

| To... | Use... |
|---|---|
| Load a test configuration from the resources library | loadConfigFromResourcesLibrary |
| Save a test configuration to the resources library | saveConfigToResourcesLibrary |

## createApp

Creates and returns a new application with the given characteristics.

### Return Value

The new application, as a WebObject.

### Syntax

```
createApp(script, ratio)
```

### Parameters

| | |
|---|---|
| `script` | The application to use when running tests, obtained using `getApplicationScriptFromResourcesLibrary` |
| `ratio` | The proportion of traffic that the application is assigned within the application mix |

## createApplicationMix

Creates and returns a new application mix with the given characteristics.

### Return Value

The new application mix, as a WebObject

### Syntax

```
createApplicationMix(name, distributionType, noUsers, direction, topology)
```

### Parameters

| | |
|---|---|
| `name` | The name of the new application mix |

| | |
|---|---|
| `distributionType` | One of the following:<br>• "USERS"<br>• "THROUGHPUT"<br>See the *IxChariot User Guide* for the characteristics of each distribution type. |
| `noUsers` | Overall number of users for the application mix |
| `direction` | One of:<br>• "SRC_TO_DEST" – for traffic flowing from source to destination<br>• "DEST_TO_SRC" – for traffic flowing from destination to source |
| `topology` | One of:<br>• "FULL_MESH" – for creating full-mesh connections between source and destination endpoints<br>• "ROUND_ROBIN" – for creating one-to-one connections between source and destination endpoints |

## createEndpoint

Creates and returns an endpoint with the given characteristics.

### Return Value

The new endpoint, as a WebObject.

### Syntax

```
createEndpoint(testIP, mgmtIP)
```

### Parameters

| | |
|---|---|
| `testIP` | The IP to be used for test traffic for the new endpoint |
| `mgmtIP` | The IP to be used for management traffic for the new endpoint |

## createFlow

Creates and returns a new flow with the given characteristics.

### Return Value

The new flow, as a WebObject

### Syntax

```
createFlow(script, users, protocol, sourceQoS, destinationQoS)
```

### Parameters

| | |
|---|---|
| `script` | The flow to use for running tests, obtained using `getFlowScriptFromResourcesLibrary` |
| `users` | The number of users for the flow |
| `protocol` | One:<br>• "TCP"<br>• "UDP"<br>• "RTP"<br>Depending on the flow, some of these options may not be valid. For example, for a UDP Baseline Performance flow, you cannot set TCP. |
| `sourceQoS` | The QoS template to use in the test for the source endpoint, obtained using `getQoSTemplateFromResourcesLibrary` |
| `destinationQoS` | The QoS template to use in the test for the destination endpoint, obtained using `getQoSTemplateFromResourcesLibrary` |

## createFlowGroup

Creates and returns a flow group with the given characteristics.

### Return Value

The new flow group, as a WebObject.

### Syntax

```
createFlowGroup(name, direction, topology)
```

### Parameters

| | |
|---|---|
| `name` | The name of the new flow group |
| `direction` | One of the following: <br>• "SRC_TO_DEST" – for traffic flowing from source to destination<br>• "DEST_TO_SRC" – for traffic flowing from destination to source<br>• "BOTH" – for traffic flowing in both directions |
| `topology` | One of the following: <br>• "FULL_MESH" – for creating full mesh connections between source and destination endpoints<br>• "ROUND_ROBIN" – for creating one-to-one connections between source and destination endpoints |

## createMulticastFlow

Creates and returns a new multicast flow with the given characteristics.

### Return Value

The new multicast flow, as a WebObject.

### Syntax

```
createMulticastFlow(script, multicastAddrPort, protocol, sourceQoS)
```

### Parameters

| | |
|---|---|
| `script` | The flow to use when running tests, obtained using `getFlowScriptFromRe-sourcesLibrary` |
| `multicastAddrPort` | The multicast address and port, in the address:port format |
| `protocol` | One of: <br>• "UDP" <br>• "RTP" |
| `sourceQoS` | The QoS template to use in the test for the source endpoint, obtained using `getQoSTemplateFromResourcesLibrary` |

## createMulticastGroup

Creates and returns a multicast group with the given name. The topology for the new group will be set to "MULTICAST" and the direction of the traffic will be "SRC_TO_DEST".

### Return Value

The new multicast group, as a WebObject

### Syntax

```
createMulticastGroup(name)
```

### Parameters

| | |
|---|---|
| `name` | The name of the new multicast group |

## changeFlowScriptParameterValue

Modifies the value of a flow parameter

### Syntax

```
changeFlowScriptParameterValue(flowScript, parameterName, parameterValue)
```

### Parameters

| | |
|---|---|
| `flowScript` | The flow whose parameter to modify, obtained using `getFlowScriptFromResourcesLibrary` |
| `parameterName` | The flow parameter to modify |
| `parameterValue` | The parameter value to modify |

## deleteOldestTestResults

Deletes test results from the resources library for the specified user.

NOTE Depending on the number of test results, the process might take up to several minutes.

### Return Value

None

### Syntax

```
deleteOldestTestResults(apiConnection, userName, howManyTestsToDelete)
```

### Parameters

| | |
|---|---|
| `apiConnection` | The object returned by webApi.connect() |
| `userName` | The user name for which to delete the test results |
| `howManyTestsToDelete` | How many test results to delete (starting from older to newer) |

## exportConfigFromResourcesLibraryToFile

Exports a test configuration from the resources library to a file.

### Return Value

None

### Syntax

```
exportConfigFromResourcesLibraryToFile(session, configName, filePath)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `configName` | The name of the configuration to export. The configuration must already exist in the resources library. |
| `filePath` | The path and filename where to export the configuration. The file must have the **.ixcfg** extension. |

## getAppLevelResults

Retrieves application-level results for the specified statistics. Can be used during the test run or after the test has ended. Will return all available results since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getAppLevelResults(session, stats, group, app)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The application mix that contains the application |
| `app` | The application for which to retrieve the results |

## getApplicationScriptFromResourcesLibrary

Looks up the requested application in the resource library and returns it.

### Return Value

The requested application, or 'None' if the application was not found.

### Syntax

```
getApplicationScriptFromResourcesLibrary(session, applicationName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `applicationName` | The name of the application to retrieve from the resources library |

## getEndpointFromResourcesLibrary

Looks up the requested endpoint in the resource library and returns it.

### Return Value

The requested endpoint, or 'None' if the endpoint was not found.

### Syntax

```
getEndpointFromResourcesLibrary(session, endpointName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `endpointName` | The name of the endpoint to retrieve from the resources library |

## getFlowLevelResults

Retrieves flow-level results for the specified statistics. Can be used during the test run or after the test has ended. Will return all available results since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getFlowLevelResults(session, stats, group, flow, protocol)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The flow group/multicast group that contains the flow |
| `flow` | The flow for which to retrieve the results |
| `protocol` | The test protocol for the flow. One of :<br>• "TCP"<br>• "UDP"<br>• "RTP" |

## getFlowScriptFromResourcesLibrary

Looks up the requested flow in the resource library and returns it.

### Return Value

The requested flow, or 'None' if the flow was not found.

### Syntax

```
getFlowScriptFromResourcesLibrary(session, flowName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `flowName` | The name of the flow to retrieve from the resources library |

## getGroupLevelResults

Retrieves group-level results for the specified statistics. Can be used during the test run or after the test has ended. Will return all available results since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getGroupLevelResults(session, stats, group)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. |
| `group` | The application mix/flow group/multicast group for which to retrieve the statistics. The names should be identical to those in the results CSV. |

## getQoSTemplateFromResourcesLibrary

Looks up the requested QoS template in the resource library and returns it.

### Return Value

The requested QoS template, or 'None' if the template was not found.

### Syntax

```
getQoSTemplateFromResourcesLibrary (session, qosTemplateName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `qosTemplateName` | The name of the QoS template to retrieve from the resources library |

## getTestLevelResults

Retrieves test-level results for the specified statistics. Can be used during the test run or after the test has ended. Will return all available results, since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getTestLevelResults(session, stats)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |

## getUserLevelResultsFromApp

Gets user-level results for the specified statistics, when the user is running an application within an application mix. Can be used during the test run or after the test has ended. Will return all available results since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getUserLevelResultsFromApp(session, stats, group, app, user)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The application mix that contains the application |
| `app` | The application that contains the user |
| `user` | The user from the application for which to retrieve the results. The number between 1 and the number of users running the application in the application mix |

## getUserLevelResultsFromFlow

Gets user-level results for the specified statistics, when the user is running a flow within a flow group/multicast group. Can be used during the test run or after the test has ended. Will return all available results since the beginning of the test.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list of StatisticValue objects.

Each StatisticValue object contains a statistic value at a specific timestamp. This function will return all the available statistic values collected since the beginning of the test. For details, see the definition of the Statistic and StatisticValue classes.

### Syntax

```
getUserLevelResultsFromFlow(session, stats, group, flow, protocol, user)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The flow group/multicast group that contains the flow |
| `flow` | The flow for which to get the results |
| `protocol` | The test protocol for the flow. One of: <br>• "TCP" <br>• "UDP" <br>• "RTP" |
| `user` | The user from the flow for which to retrieve the results. The number between 1 and the number of users running the flow within the flow group/multicast group. |

## getUserInfoFromApp

Gets user info for the specified statistics (e.g. Name, Location, Version, Operating System, IP addresses and errors for the source and destination endpoints). Can be used during the test run or after the test has ended.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list with one StatisticValue object.

The StatisticValue object contains a statistic value at time stamp 0. For details, see the definition of the Statistic and StatisticValue classes.

Unlike the other statistics, the User Info results are not reported per time stamp. Time stamp 0 is arbitrarily used to keep the same API as for the other statistics. During the test execution, some User Info results might be added or updated (by overwriting the previous values).

### Syntax

```
getUserInfoFromApp(session, stats, group, app, user)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded. |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The application mix that contains the application. |
| `app` | The application that contains the user. |
| `user` | The user from the application for which to retrieve the results. The number between 1 and the number of users running the application in the application mix. |

## getUserInfoFromFlow

Gets user info for the specified statistics (e.g. Name, Location, Version, Operating System, IP addresses and errors for the source and destination endpoints). Can be used during the test run or after the test has ended.

### Return Value

A list of Statistic objects (one object for each requested statistic). Each Statistic object contains a list with one StatisticValue object.

The StatisticValue object contains a statistic value at time stamp 0. For details, see the definition of the Statistic and StatisticValue classes.

Unlike the other statistics, the User Info results are not reported per time stamp. Time stamp 0 is arbitrarily used to keep the same API as for the other statistics. During the test execution, some User Info results might be added or updated (by overwriting the previous values).

### Syntax

```
getUserInfoFromFlow(session, stats, group, flow, protocol, user)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session where the test is running/loaded. |
| `stats` | The list of statistics names for which to get the values. The names should be identical to those in the results CSV. |
| `group` | The flow group/multicast group that contains the flow. |
| `flow` | The flow for which to get the results. |
| `protocol` | The test protocol for the flow. |
| `user` | The user from the flow for which to retrieve the results. The number between 1 and the number of users running the flow in the flow group/multicast group. |

## importConfigFromFileToResourcesLibrary

Imports a configuration from a file to the resources library.

### Return Value

The name of the imported configuration. The names of the configurations in the resources library must be unique. As such, if a name conflict is detected, the name of the current configuration will be changed to ensure uniqueness. In this case, the name returned by the function will be different from the original name of the configuration.

### Syntax

```
importConfigFromFileToResourcesLibrary(session, filePath)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| filePath | The path to a previously exported configuration. The file must have the **.ixcfg** extension for configurations exported from IxChariot 9.3 or newer or the **.zip** extension for configurations exported from IxChariot 9.2 or older (up to 8.0 included). |

## loadConfigFromResourcesLibrary

Loads a test configuration from the resources library into the current session.

### Return Value

None

### Syntax

```
loadConfigFromResourcesLibrary(session, configName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `configName` | The name of the configuration to load |

## saveConfigToResourcesLibrary

Saves a test configuration to the resources library, to allow access at a later time.

### Return Value

None

### Syntax

```
saveConfigToResourcesLibrary(session, configName)
```

### Parameters

| | |
|---|---|
| `session` | The current IxChariot session |
| `configName` | The name under which to save the configuration in the resources library |

## Session Methods

Sessions have the following methods:

| To... | Use... |
|---|---|
| Check the notification queue | checkNotifications |
| Collects debug diagnostics | collectDiagnosticsToFile |
| Delete a configuration | deleteConfiguration |
| Delete a session | deleteSession |
| Export a configuration to a file | exportConfigurationToFile |
| Get the description of a configuration | findConfigurationByName |
| Get the list of saved test configurations | getConfigurations |
| Get the list of notifications | getNotifications |
| Update the properties of the session | httpRefresh |
| Import a configuration | importConfigurationFromFile |
| Load a configuration | loadConfiguration |
| Run the current configuration and wait for it to finish | runTest |

| To... | Use... |
|---|---|
| Save the current configuration | saveConfiguration |
| Start a session | startSession |
| Start running the current configuration without waiting for it to finish | startTest |
| Stop a session | stopSession |
| Stop the current running test | stopTest |
| Wait for the current test to stop, when it was started with **startTest** | waitTestStopped |

## checkNotifications

Checks the notification queue and throws a WebException if it finds any errors. A WebException is a sub-class of the Python Exception class that only the WebAPI throws.

### Return Value

WebException on error

### Syntax

```
checkNotifications()
```

### Parameters

None

## collectDiagnosticsToFile

Collects debug diagnostics (potentially, a long process) and delivers them to the file handle or file-like object `diagFile`. `diagFile` must have been opened in binary mode. The contents written to the file are for Ixia's internal use and subject to change without notice, so scripts should not attempt to inspect or manipulate the file contents.

### Return Value

None

### Syntax

```
collectDiagnosticsToFile(diagFile)
```

### Parameters

| | |
|---|---|
| `diagFile` | File to store diagnostics in. |

## deleteConfiguration

Deletes a configuration

### Return Value

None

### Syntax

```
deleteConfiguration(name)
```

### Parameters

| | |
|---|---|
| `name` | Configuration to delete. |

## deleteSession

Stops a session if it is still running, and then deletes it from the server.

NOTE Deleting the Python Session object does not cause the session on the server to be deleted. You must call `deleteSession` to delete a session.

### Return Value

None

### Syntax

```
deleteSession()
```

### Parameters

None

## findConfigurationByName

Returns a WebObject (same as the objects in the list returned by `getConfigurations`) containing the description of the configuration passed in.

### Return Value

WebObject

### Syntax

`findConfigurationByName(name)`

### Parameters

| | |
|---|---|
| `name` | Configuration to retrieve the description for. |

## getConfigurations

Returns a list of WebObjects representing the test configurations saved on the server. Each object in the list has at least the following fields:

| | |
|---|---|
| `id` | Configuration ID |
| `name` | Name of the configuration |
| `description` | Description of the configuration |

### Return Value

List of WebObjects

### Syntax

`getConfigurations()`

### Parameters

None

## getNotifications

Retrieves the list of notifications. The list is cleared when `startTest()` is called.

### Return Value

Notification list

### Syntax

`getNotifications()`

### Parameters

None

## httpRefresh

Updates the session properties, as described below, from the server. Any script that loops waiting for a property to change must call `httpRefresh` -- otherwise the property will never change.

### Return Value

None

### Syntax

```
httpRefresh()
```

### Properties

| Property Name | Type | Description |
|---|---|---|
| `sessionId` | Integer | Session ID. |
| `sessionType` | String | Type of session. |
| `testIsRunning` | Boolean | `true` if the test is running, else `false`. |
| `testConfigName` | String | Name of the test configuration where the current configuration has been saved to or loaded from. |
| `state` | String | Session state. Possible values are:<br>`Initial`<br>`Starting`<br>`Active`<br>`Stopping`<br>`Stopped`<br>`Dead` |
| `subState` | String | Session substate. Values vary based on `sessionType` and session `state`.<br>For example, an Active session state may show substates that reflect the test controller state (such as Unconfigured, Configuring, Running, etc.). |
| `creationDate` | String | Date and time, in ISO 8601 format, that the session was created. |

| | | |
|---|---|---|
| `creationTime` | Integer | Epoch time (seconds since 1/1/1970) when the session was created. |
| `startingTime` | Integer | Epoch time (seconds since 1/1/1970) when the session was started. |
| `elapsedTime` | Integer | Number of seconds elapsed since the session was started. |
| `stoppingTime` | Integer | Epoch time (seconds since 1/1/1970) when the session was stopped. |

## runTest

Runs the current configuration. Nothing is returned until the test run has completed. The result contains, at a minimum, a field named `id` with the `testId` of the test run (as might be required for other method calls like `getStatsCsvToFile`).

### Return Value

WebObject

### Syntax

```
runTest()
```

### Parameters

None

## startSession

Starts the session and waits for it to finish starting. While it is starting, the session is in the *Starting* state. When it is finished starting, it is in the *Active* state. Sessions must be in the *Active* state to run tests.

### Return Value

None

### Syntax

```
startSession()
```

### Parameters

None

## startTest

Starts running the current configuration and returns immediately. You can use the property `testIsRunning` to determine if the test is still running after calling `startTest`. `startTest` returns the same WebObject as described under `runTest()`.

### Return Value

WebObject

### Syntax

`startTest()`

### Parameters

None

## stopSession

Stops the session, and waits for it to enter the *Stopped* state.

### Return Value

None

### Syntax

`stopSession()`

### Parameters

None

## stopTest

Stops the current running test.

### Return Value

None

### Syntax

`stopTest()`

### Parameters

None

## waitTestStopped

Waits for the current test to stop (when the test was started with **startTest**) and then checks for any error notifications. If it finds any error notifications, it throws a WebException.

### Return Value

WebException if errors found.

### Syntax

```
waitTestStopped()
```

### Parameters

None

## Statistic Class

Describes a statistic from the test.

### Members

| | |
|---|---|
| `session` | The current IxChariot session |
| `values` | The list representing values for the statistic over time, containing elements of the `StatisticValue` type |

### Functions

| | |
|---|---|
| `add_value (value)` | Takes a `StatisticValue` as parameter, appends it to the list of values. |

## StatisticValue Class

Describes a value for a statistic at a moment in time.

| | |
|---|---|
| `timestamp` | The moment in the test when the value was recorded (in milliseconds) |
| `value` | The actual value recorded |

## WebObjects

Many APIs served by a ReSTful server return (and accept) objects as either an XML-or JSON-formatted text. The IxChariot server uses JSON. To make it easier to work with JSON, the `ixia.webapi` module includes a Python WebObject class that wraps the JSON text and allows for a simple object-like notation that is much simpler to type and manipulate than raw JSON strings.

For example, JSON objects are typically stored as a series of nested dictionaries and lists. A list of cars and their owners' names could be represented by the following dictionary:

```
carList = [{"model": "Ford", "year":2008, "priorOwners":[{"name": "Bill"}, {"name": "Mary"}]},
```

```
{"model": "Toyota", "year":2010, "priorOwners":[{"name": "Ida"}, {"name": "Frank"}]}]
```

When a car is sold, a new prior owner needs to be added, so the way to do this unassisted is:

```
carList[1]["priorOwners"].append({"name": "Ionut"})
```

This is a simple example, but the properties could hold other lists and dictionaries nested to an arbitrary depth. This quickly becomes unreadable. With a WebObject, the syntax becomes much easier to script:

```
carList[0].priorOwners.append(name="Ionut")
```

For more examples of manipulating web objects, review the sample scripts. Note that the properties are created dynamically to match the server response, so they are not documented here (it is specific to the web service and the WebAPI library has no service-specific code).