Clarius Automation Framework API and SDK

Programming Guide

Version 2.0.0

Register now!
Click the following link to protect your product.
tek.com/register

077-1854-00 December 2024

Tektronix, Inc.
14150 SW Karl Braun Drive
P.O. Box 500
Beaverton, OR 97077
US

For product information, sales, service, and technical support visit *tek.com* to find contacts in your area. For warranty information visit *tek.com/warranty*.

Contents

Welcome

The Clarius automation framework programming guide offers details about the REST APIs and SDK. It covers instructions on how to use these resources, authenticate, and construct REST API and SDK calls.

Intended audience

The information in this guide is intended for administrators and programmers who want to use the REST APIs and SDK to configure and manage the Clarius automation framework.

Understanding the APIs and SDK

You can use the APIs and SDK to build interactive clients of the Clarius automation framework. The REST API and SDK are available for all licensed users.

Clarius automation framework APIs communicate with the server over HTTP, exchanging representations of Clarius objects. These representations take the form of JSON elements.

Getting help and support

Product documents

Use the product documents for more information about getting started with the Clarius, the application functions, and how to remotely use the application.

Table 1: Clarius automation framework and application documents

To learn about	Use this document
How to install the Clarius	Clarius Automation Framework Getting Started Guide
How to use the application	Clarius Compliance Application Help
How to automate using the API and SDK commands	Clarius Automation Framework (API and SDK) Programming Guide

Conventions

This application help uses the following conventions:

- The terms "Application" and "Software" refer to the Clarius compliance application.
- The term "target system" refers to the Computer/Laptop where the Clarius automation framework and application is installed.
- The acronym “DUT” is an abbreviation for Device Under Test.
- The term “select” refers to the two methods of choosing a screen item (button control or list item): using a mouse or using the touch screen.
- A **Note** identifies important information.
- The acronym "Tx""Rx" is an abbreviation for TransmitterReceiver.

Technical support

Tektronix values your feedback on our products. To help us serve you better, please send us your suggestions, ideas, or comments on your application or oscilloscope. Contact Tektronix through mail, telephone, or website. See *Contacting Tektronix* for more information.

When you contact Tektronix Technical Support, please include the following information (be as specific as possible):

General information

- All instrument model numbers
- Hardware options, if any
- Modules used
- Your name, company, mailing address, phone number, FAX number
- Please indicate if you would like to be contacted by Tektronix about your suggestions or comments.

Application specific information

- Software version number
- Description of the problem

- If possible, save the log file(s) and share it with the Tektronix support person to understand the problem and get it resolved.

Clarius API Programming

Introduction

You can use the REST API to communicate remotely with the Clarius automation framework. The flexibility and scalability of REST API make it an excellent choice for integrating the Clarius automation framework into your applications and performing complex operations on a large scale. By accessing the APIs, you can integrate the Clarius automation framework into your applications and perform operations as per your requirements.

The REST API enables you to create, update, and search data in the Clarius automation framework by sending HTTPS requests to endpoints. You can access and use several sets of information or resources, based on the requests sent to the framework. Resources include records, query results, metadata, and more.

REST API uses RESTful architecture to provide a straightforward and consistent interface. A primary benefit of REST API is that it doesn't require much tooling to access your data. It is simpler to use and provides many functionality. However, understanding and using REST API requires basic familiarity with software development, web services, and the Clarius automation framework user interface.

API function changelogs

This section lists the newly added API functions, modifications in the existing API functions and deprecated API functions from previous release.

Newly added functions

- *Delete executed tests* on page 44
- *Delete waveforms of executed test* on page 73

Changes in existing functions

Initialization	If the SSL certificate port and API port is changed during the installation of the Clarius automation framework, these ports must be configured during the initialization of Clarius SDK.
Get list of application on page 19	JSON structure updated with additional parameters.
Get application by id on page 20	
Test bench on page 29	JSON structure updated with additional parameters.
Run a test on page 37	JSON structure update to include additional fields for acquisition mode and recorded waveform path
Merge the test session on page 44	
Create a test execution draft on page 47	
Update the test execution draft on page 50	
Merge test execution draft on page 54	
Get test execution status by id on page 65	
Get test execution result on page 69	
Test sequence on page 77	

Deprecated functions

No functions are deprecated.

Send requests

Use the following guidelines when sending requests using the REST API.

URL structure

The Clarius API follows the below structure. The arguments passed with the URL are specified within double angular brackets.

`https://<<host ip>>:<<port id>>/clarius/<<arguments>>`

These arguments can be both mandatory and optional. The mandatory arguments are listed in **bold** font.

Supported methods

You can perform basic CRUD operations (create, retrieve, update, and delete) on Clarius using standard HTTP method requests, as summarized in the following table.

HTTP method	CRUD operation	Description
POST	Create	Create a REST API resource
GET	Retrieve	Retrieve information about the REST API resource
PUT	Update/Replace	Update or Replace a REST API resource
DELETE	Delete	Delete a REST API resource or related component

Media type

The following media type is supported by the REST API.

- application/json

Status codes

When you call any of the REST resources, the Response header returns one of the standard HTTP status codes defined in the following table.

HTTP Status code	Description
200 OK	The request was successfully completed. A `200` status is returned for a successful `GET` or `POST` method.

Table continued…

HTTP Status code	Description
201 Created	The request has been fulfilled and resulted in a new resource being created. The response includes a location header containing the canonical URI for the newly created resource. A `201` status is returned from a synchronous resource creation or an asynchronous resource creation that was completed before the response was returned.
204 No Content	The request is fulfilled, but there is no content available in the system.
400 Bad Request	The request could not be processed because it contains missing or invalid information (such as a validation error on an input field or a missing required value).
401 Unauthorized	The request is not authorized. The authentication credentials included with this request are missing or invalid.
404 Not Found	The request includes a resource URI that does not exist.
405 Method Not Allowed	The HTTP verb specified in the request (DELETE, GET, POST, PUT) is not supported for this request URI.
409 Conflict	The request could not be completed due to a conflict with the current state of the resource. Either the version number does not match, or a duplicate resource was requested.
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.

Error messages

All REST APIs return JSON output appropriate for the API invoked. HTTP Status codes other than 200 are used as appropriate to indicate various failures, along with JSON for detailed error messages.

Generate access token

This command generates a new access token. The access token generated will be valid for a time period (7200 seconds) only and upon the expiry of the access_token, you must refresh the access token by sending a refresh_token request to the server.

POST `https://`**<<host ip>>:<<portid>>**`/clarius/oauth2/token`

Request

Header	-

Body	<pre>{
	"grant_type": "",
	"client_id": "",
	"client_secret": "",
	"username": ,
	"password": ,
	}</pre>

Response

On successful verification of the user credentials, you will receive access_token, expires_in, token_type, and refresh_token in JSON format.

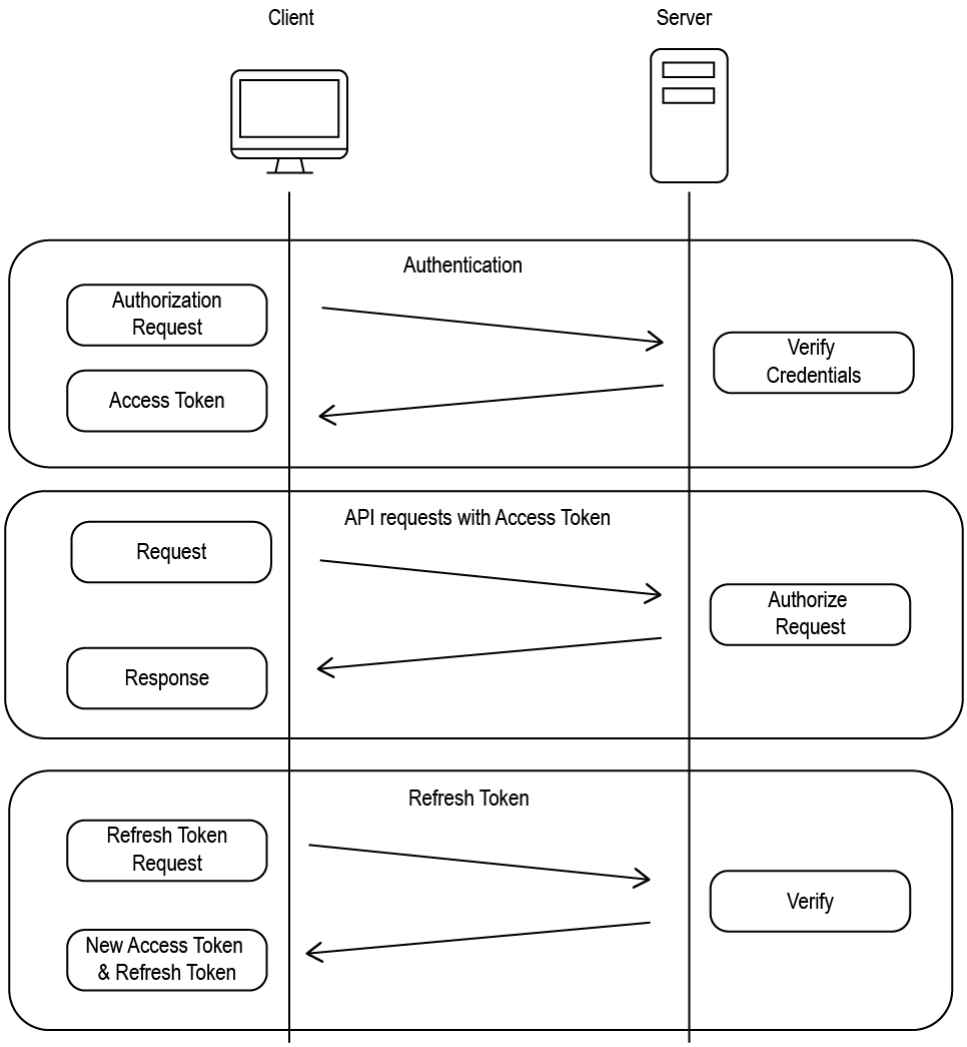
Table 2: Status codes

Status code	Description

200	License key retrieved successfully.
	
	{
	"access_token": "",
	"expires_in": ,
	"token_type": "",
	"refresh_token": ""
	}

404	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

Clarius API session



Licensing

The licensing API contains the set of rest endpoints that are used to get the host ID of the system, activating and deactivating the application license, get the details of all installed licenses, and get the list of active applications.

Get the license keys

This command queries the license key.

```
GET https://<<host ip>>:<<portid>>/clarius/license/$getkey
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and license keys in JSON format.

Table 3: Status codes

Status code	Description

200	License key retrieved successfully.
	
	{
	"deploymentId": ""
	}

404	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

Get the active applications and license details

This command retrieves the list of all active applications and their license validity details.

GET `https://<<host ip>>:<<portid>>/clarius/license`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and license information in JSON format.

Table 4: Status codes

Status code	Description

200	License validity of all application ids are retrieved successfully.
	
	{
	"id": "",
	"type": "",
	"application": "",
	"licenseExpired": true | false,
	"maintenanceExpired": true | false,
	"expirationDateGMT": "DD/MM/YYYY-HH:MM:SS",
	"buildExpirationDateGMT": "DD/MM/YYYY-HH:MM:SS"
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

404	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

Get license details

This command retrieves all active license details such as license type, expiry date, linked application ids, etc.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/license/$details`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and list of active license details in JSON format.

Table 5: Status codes

Status code	Description

200	Active license details were retrieved successfully.
	
	{
	"description": "",
	"id": "",
	"nomenclature": "",
	"type": "",
	"checkedOutDateGMT": "DD/MM/YYYY-HH:MM:SS",
	"expirationDateGMT": "DD/MM/YYYY-HH:MM:SS",
	"buildExpirationDateGMT": "DD/MM/YYYY-HH:MM:SS",
	"checkedOut": "",
	"applications": [""]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

404	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

Activate application license

This command activates the application license from the license file.

POST `https://<<host ip>>:<<portid>>/clarius/license/$activate`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	License file content as plain text.

Response

Returns command execution status.

Table 6: Status codes

Status code	Description
200	Application license activated successfully.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Deactivate application license

This command deactivates the application license by license Id.

POST `https://`**<<host ip>>:<<portid>>**`/clarius/license/$deactivate/`**<<licenseId>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
licenseId	Application license id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status.

Table 7: Status codes

Status code	Description

200	Application license deactivated successfully.
	
	{
	"exitfile": ""
	}
	
	Note: Save the response data with a .lic extension file and share it with Tektronix Application engineer.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Application

The application API contains the set of rest endpoints that are used for creating, retrieving, updating, and deleting the application from the database. It also has an additional endpoint to list down all the applications present in the database based on the query parameters.

Get list of application

This command retrieves the list of application(s).

```
GET https://<<host ip>>:<<portid>>/clarius/application?
techCategoryType=<<Type>>&techCategorySubType=<<SubType>>&testCategoryType=<<Type>>
&testCategorySubType=<<SubType>>&name=<<Applicationname>>&mode=<<Executionmode>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Table continued…

Name	Description
techCategoryType	Type of technology category
techCategorySubType	Subtype of technology category
testCategoryType	Type of test category
testCategorySubType	Subtype of test category
name	Name of the application
mode	Test execution mode

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and list of application(s) in JSON format.

Table 8: Status codes

Status code	Description

200	Application list retrieved successfully.
	
	[
	{
	"id":"",
	"name":"",
	"description":"",
	"technologyCategory":{
	"type":"",
	"subType":""
	},
	"testCategory":{
	"type":"",
	"subType":""
	},
	"licenseActive":"true | false",
	"executionMode":"COMPLIANCE | USER | CHARACTERIZATION",
	"version":""
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get application by id

This command retrieves complete information of the application by its id.

GET https://**<<host ip>>:<<portid>>**/clarius/application/**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Application id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and application details by id in JSON format.

Table 9: Status codes

Status code	Description

200	Application retrieved successfully.
	
	{
	"id": "",
	"name": "",
	"displayName": "",
	"technologyCategory": {
	"type": "",
	"subType": ""
	},
	"testCategory": {
	"type": "",
	"subType": ""
	},
	"executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
	"description": "",
	"version": "",
	"settings": [
	{
	"name": "",
	"type": "",
	"displayName": "",
	"group": "",
	"reference": [],
	"referenceType": "",
	"value": {},
	"constraints": "",
	"unit": "",
	"description": "",

Table continued…

Status code	Description

	"additionalProperties": {},
	"referenceGroupBy": {},
	"scenarioName": "",
	"stepName": "",
	"category": {},
	"tag": "",
	"global": true | false,
	"mandatory": true | false,
	"editable": true | false,
	"deprecated": true | false,
	"rules": []
	}
	],
	"instrumentDefinitions": [
	{
	"name": "",
	"type": "",
	"mandatory": true | false,
	"constraints": ""
	}
	],
	"sourceDefinitions": [
	{
	"name": "",
	"type": "",
	"range": {},
	"signals": []
	}
	],
	"scenarios": [
	{
	"id": "",
	"name": "",
	"displayName": "",
	"category": {},
	"description": "",
	"settings": [],
	"steps": [],
	"dependencies": [],
	"dependents": true | false
	}
	],
	"licenseActive": true | false
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
403	Requested resource is forbidden and the application is not licensed.
404	The endpoint cannot be reached.

Table continued…

Status code	Description
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Limits data

The limits API contains the set of rest endpoints that are used for creating, retrieving, updating, and deleting analysis result limits from the database.

Get all limits data

This command retrieves all limits data present in the database.

```
GET https://<<host ip>>:<<portid>>/clarius/limits
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and limits details in JSON format.

Table 10: Status codes

Status code	Description

200	Limits data list retrieved successfully.
	{
	"id": "",
	"description": "",
	"limits": [
	{
	"name": "",
	"displayName": "",
	"group": "",
	"idealValue": 0,
	"lowLimit": {
	"value": 0,
	"comparator": ""
	},
	"highLimit": {
	"value": 0,
	"comparator": ""
	},
	"unit": "string",
	"additionalInfo": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get limits data by id

This command retrieves the complete details of the limits data from the database.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/limits/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Limits data id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and limits details in JSON format.

Table 11: Status codes

Status code	Description

200	Limits data retrieved successfully.
	{
	"id": "",
	"description": "",
	"limits": [
	{
	"name": "",
	"displayName": "",
	"group": "",
	"idealValue": 0,
	"lowLimit": {
	"value": 0,
	"comparator": ""
	},
	"highLimit": {
	"value": 0,
	"comparator": ""
	},
	"unit": "",
	"additionalInfo": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Update limits data by id

This command updates the limits data for the given id.

PUT `https://`**<<host ip>>:<<portid>>**`/clarius/limits/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Limits data id

Request

Header	Authorization: {Bearer access token}

Body	{
	"id": "",
	"description": "",
	"limits": [
	{
	"name": "",
	"displayName": "",
	"group": "",
	"idealValue": 0,
	"lowLimit": {
	"value": 0,
	"comparator": ""
	},
	"highLimit": {
	"value": 0,
	"comparator": ""
	},
	"unit": "",
	"additionalInfo": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	Note: The arguments in bold font are mandatory.

Response

Returns command execution status and updates limits data.

Table 12: Status codes

Status code	Description
200	Limits data updated successfully.

Table continued…

Status code	Description

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Create an edited copy of limits data

This command creates an edited copy of limits data.

POST `https://`**<<host ip>>:<<portid>>**`/clarius/limits/edit`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Table continued…

Body	{
	"id": "",
	"description": "",
	"limits": [
	{
	"name": "",
	"displayName": "",
	"group": "",
	"idealValue": 0,
	"lowLimit": {
	"value": 0,
	"comparator": ""
	},
	"highLimit": {
	"value": 0,
	"comparator": ""
	},
	"unit": "",
	"additionalInfo": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	Note: The arguments in bold font are mandatory.

Response

Returns command execution status.

Table 13: Status codes

Status code	Description
200	Edited version of limits created successfully.

400	The validation of the request failed.
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

Table continued…

Status code	Description
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Test bench

The test bench API contains the set of rest endpoints that are used for creating, retrieving, updating, and deleting the test bench from the database. It also has an additional endpoint to list down all the test bench present in the database based on the query parameters.

Create a new test bench

This command creates a new test bench.

POST `https://<<host ip>>:<<portid>>/clarius/testbench`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	{ "id": "string", "name": "string", "description": "string", "internal": bool, "validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA" "acquisitionMode": "LIVE", "technologies": ["string"], "applications": ["string"], "hubAddress": "http://<ip address>:18000", "instruments": [{ "id": "string", "name": "string", "category": "string", "address": "string",

```
      "description": "string",
      "properties": {
        "additionalProp1": {},
        "additionalProp2": {},
        "additionalProp3": {}
      },
      "extensions": [
        {
          "id": "string",
          "name": "string",
          "category": "string",
          "address": "string",
          "description": "string",
          "properties": {
            "additionalProp1": {},
            "additionalProp2": {},
            "additionalProp3": {}
          }
        }
      ]
    }
  ],
  "availability": "AVAILABLE"
}
```


Note: The arguments in bold font are mandatory.

Table 14: Instrument details

Type	Category
SIGNAL_ANALYZER	Real Time Scope
SIGNAL_GENERATOR	Arbitrary Waveform Generator
	Arbitrary Function Generator
	Bit Error Rate Tester
CUSTOM	Custom Instrument

Response

Returns command execution status and creates a new test bench.

Table 15: Status codes

Status code	Description
200	Test bench created successfully.

Table continued…

Status code	Description

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of test benches

This command retrieves the list of test benches for the given technology and name.

```
GET https://<<host ip>>:<<portid>>/clarius/testbench?
technology=<<technology>>&name=<<name>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
technology	Technology name
name	Test bench name
application	Application name
acquisitionMode	Acquisition mode
availability	Availability status of the test bench

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and list of test benches in JSON format.

Table 16: Status codes

Status code	Description

200	Test bench list retrieved successfully.
	
	[
	{
	"id": "string",
	"name": "string",
	"description": "string",
	"internal": bool,
	"validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA"
	"acquisitionMode": "LIVE",
	"technologies": [
	"string"
	],
	"applications": [
	"string"
	],
	"hubAddress": "string",
	"instruments": [
	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"extensions": [
	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"availability": "AVAILABLE"
	}
	]

Table continued…

Status code	Description
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test bench details by id

This command retrieves the test bench details by its id.

GET https://**<<host ip>>:<<portid>>**/clarius/testbench/**<<identifier>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
identifier	Test bench id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test bench details in JSON format.

Table 17: Status codes

Status code	Description
200	Test bench retrieved successfully. { "id": "string", "name": "string", "description": "string", "internal": bool, "validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA" "acquisitionMode": "LIVE", "technologies": ["string"], "applications": ["string"], "hubAddress": "string", "instruments": [

Table continued…

Status code	Description

	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"extensions": [
	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"availability": "AVAILABLE"
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Update the test bench

This command updates the test bench details by its id.

PUT `https://`**<<host ip>>:<<portid>>**`/clarius/testbench/`**<<identifier>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
identifier	Test bench id

Request

Header	Authorization: {Bearer access token}

Body	{
	"id": "string",
	"name": "string",
	"description": "string",
	"internal": bool,
	"validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA"
	"acquisitionMode": "LIVE",
	"technologies": [
	"string"
	],
	"applications": [
	"string"
	],
	"hubAddress": "string",
	"instruments": [
	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"extensions": [
	{
	"id": "string",
	"name": "string",
	"category": "string",
	"address": "string",
	"description": "string",
	"properties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"availability": "AVAILABLE"
	}
	Note: The arguments in bold font are mandatory.

Response

Returns command execution status and updates the test bench.

Table 18: Status codes

Status code	Description
200	Test bench updated successfully.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete the test bench

This command deletes the given test bench by its id.

DELETE `https://`**<<host ip>>:<<portid>>**`/clarius/testbench/`**<<identifier>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
identifier	Test bench id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test bench.

Table 19: Status codes

Status code	Description
204	Test bench deleted successfully.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Test execution

The Test execution API contains the set of rest endpoints that are used for running tests, getting test configurations, execution status, execution results, and take particular actions for the notifications.

Run a test

This command runs the test.

POST `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	{
	"executionName": "Test name",
	"linkId": "",
	"tags": [""],
	"description": "",
	"acquisitionMode":"LIVE|RECORDED",
	"waveformFolderPath":string
	"executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
	"testBenchId": "",
	"applicationRequests": [
	{
	"technology": "",
	"testCategory": {
	"type": "",
	"subType": ""
	},

```
"loop": {
  "sources": [
    [
      {
        "name": "",
        "type": "",
        "signals": [
          {
            "name": "",
            "category": "TRANSMITTER | RECEIVER",
            "probeMethod": "SINGLE_ENDED | DIFFERENTIAL",
            "singleEnded": [],
            "differential": []
          }
        ]
      }
    ]
  ],
  "applicationId": ""
},
"tag": "",
"settings": [
  {
    "name": "",
    "type": "",
    "displayName": "",
    "group": "",
    "reference": [],
    "referenceType": "DEFAULT | ITERATIVE | AGGREGATE",
    "value": {},
    "constraints": "",
    "unit": "",
    "description": "",
    "additionalProperties": {},
    "referenceGroupBy": {},
    "scenarioName": "",
    "stepName": "",
    "category": {
     "type": "",
     "subType": ""
    },
    "tag": "",
    "global": true | false,
    "mandatory": true | false,
    "editable": true | false,
    "deprecated": true | false,
    "rules": [
     {
      "name": "",
      "type": "FILTER | VALIDATION",
```

```
                "inputs": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                },
                "output": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                }
              }
            ]
          }
        ],
        "applicationSelections": [
          {
            "settings": [],
            "selectionId": "",
            "applicationId": "",
            "selected": true | false,
            "scenarioNames": [],
             "scenarioSelectionInfo": [
             {
           "scenarioName": "",
           "selected": true | false
             }
           ]
          }
        ]
      }
    ]
  }
```

Note: The arguments in bold font are mandatory.

Response

Returns command execution status.

Table 20: Status codes

Status code	Description
200	Application execution requests were processed successfully.

Table continued…

Status code	Description

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test execution configurations of an application

This command retrieves the test execution configurations for the given id.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and text execution configurations in JSON format.

Table 21: Status codes

Status code	Description

200	Test execution configurations were retrieved successfully.
	
	{
	"executionId": "",
	"executionName": "",
	"linkId": "",
	"tags": [""],
	"description": "",
	"acquisitionMode":"LIVE|RECORDED",
	"waveformFolderPath":string
	"executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
	"testBenchId": "",
	"applicationRequests": [
	{
	"technology": "",
	"testCategory": {
	"type": "",
	"subType": ""
	},
	"loop": {
	"sources": [
	[
	{
	"name": "",
	"type": "",
	"signals": [
	{
	"name": "",
	"category": "TRANSMITTER | RECEIVER",
	"probeMethod": "SINGLE_ENDED | DIFFERENTIAL",
	"singleEnded": [],
	"differential": []
	}
	]
	}
	]
	],
	"applicationId": ""
	},
	"tag": "",
	"settings": [
	{
	"name": "",
	"type": "",
	"displayName": "",
	"group": "",
	"reference": [],
	"referenceType": "DEFAULT",

Table continued…

Status code	Description

	"value": {},
	"constraints": "",
	"unit": "",
	"description": "",
	"additionalProperties": {},
	"referenceGroupBy": {},
	"scenarioName": "",
	"stepName": "",
	"category": {
	"type": "",
	"subType": ""
	},
	"tag": "",
	"global": true | false,
	"mandatory": true | false,
	"editable": true | false,
	"deprecated": true | false,
	"rules": [
	{
	"name": "",
	"type": "FILTER | VALIDATION",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"applicationSelections": [
	{
	"settings": [],
	"selectionId": "",
	"applicationId": "",
	"selected": true | false,
	"scenarioNames": [],
	"scenarioSelectionInfo": [
	{
	"scenarioName": "",
	"selected": true | false
	}
	]
	}

Table continued…

Status code	Description
	] }], "generateReport": true | false, "templateId": "" }
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete an executed test

This command deletes the given test execution by id.

```
DELETE https:<<host ip>>:<<portid>>/clarius/application/$execute/<<id>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test.

Table 22: Status codes

Status code	Description
204	Execution request deleted successfully.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete executed tests

This command deletes the executed tests, specified by IDs.

```
DELETE https://<<host ip>>:<<portid>>/clarius/application/$execute/tests
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	["executionId1", "executionId2", "executionId3"]

Response

Returns command execution status.

Table 23: Status codes

Status code	Description
204	Execution requests deleted successfully
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Merge the test session

This command merges the new test session with the previous session.

```
POST https://<<host ip>>:<<portid>>/clarius/application/$execute/$merge?
deleteSession=<<deleteSession>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Table continued…

Name	Description
deleteSession[1]	Boolean value determines whether to delete the previous session or not. By default the value is "false".

Request

Header	Authorization: {Bearer access token}
Body	

```
{
  "executionId": "",
  "executionName": "",
  "linkId": "",
  "tags": [""],
  "description": "",
  "acquisitionMode":"LIVE|RECORDED",
  "waveformFolderPath":string
  "executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
  "testBenchId": "",
  "applicationRequests": [
    {
      "technology": "",
      "testCategory": {
        "type": "",
        "subType": ""
      },
      "loop": {
        "sources": [
          [
            {
              "name": "",
              "type": "",
              "signals": [
                {
                  "name": "",
                  "category": "TRANSMITTER | RECEIVER",
                  "probeMethod": "SINGLE_ENDED | DIFFERENTIAL",
                  "singleEnded": [],
                  "differential": []
                }
              ]
            }
          ]
        ],
        "applicationId": ""
      },
      "tag": "",
      "settings": [
        {
          "name": "",
```

[1] This parameter is optional.

```
        "type": "",
        "displayName": "",
        "group": "",
        "reference": [],
        "referenceType": "DEFAULT",
        "value": {},
        "constraints": "",
        "unit": "",
        "description": "",
        "additionalProperties": {},
        "referenceGroupBy": {},
        "scenarioName": "",
        "stepName": "",
        "category": {
         "type": "",
         "subType": ""
       },
        "tag": "",
        "global": true | false,
        "mandatory": true | false,
        "editable": true | false,
        "deprecated": true | false,
        "rules": [
         {
          "name": "",
          "type": "FILTER | VALIDATION",
          "inputs": {
           "additionalProp1": {},
           "additionalProp2": {},
           "additionalProp3": {}
          },
          "output": {
           "additionalProp1": {},
           "additionalProp2": {},
           "additionalProp3": {}
          }
         }
       ]
      }
    ],
    "applicationSelections": [
      {
        "settings": [],
        "selectionId": "",
        "applicationId": "",
        "selected": true | false,
        "scenarioNames": [],
         "scenarioSelectionInfo": [
      {
       "scenarioName": "",
       "selected": true | false
      }
```

	] }] }], "generateReport": true | false, "templateId": "" }

Response

Returns command execution status.

Table 24: Status codes

Status code	Description
200	Application execution requests merged successfully.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Create a test execution draft

This command creates a test execution draft.

POST `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute/draft`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	

```
{
  "executionName": "",
  "linkId": "",
  "tags": [""],
  "description": "",
  "acquisitionMode":"LIVE|RECORDED",
  "waveformFolderPath":string
  "executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
  "testBenchId": "",
  "applicationRequests": [
    {
      "technology": "",
      "testCategory": {
        "type": "TX",
        "subType": "TEST"
      },
      "loop": {
        "sources": [
          [
            {
              "name": "",
              "type": "",
              "signals": [
                {
                  "name": "",
                  "category": "TRANSMITTER | RECEIVER",
                  "probeMethod": "SINGLE_ENDED | DIFFERENTIAL",
                  "singleEnded": [],
                  "differential": []
                }
              ]
            }
          ]
        ],
        "applicationId": ""
      },
      "tag": "",
      "settings": [
        {
          "name": "",
          "type": "",
          "displayName": "",
          "group": "",
          "reference": [],
          "referenceType": "DEFAULT",
          "value": {},
          "constraints": "",
          "unit": "",
```

```
            "description": "",
            "additionalProperties": {},
            "referenceGroupBy": {},
            "scenarioName": "",
            "stepName": "",
            "category": {
             "type": "",
             "subType": ""
            },
            "tag": "",
            "global": true | false,
            "mandatory": true | false,
            "editable": true | false,
            "deprecated": true | false,
            "rules": [
             {
              "name": "",
              "type": "FILTER | VALIDATION",
              "inputs": {
               "additionalProp1": {},
               "additionalProp2": {},
               "additionalProp3": {}
              },
              "output": {
               "additionalProp1": {},
               "additionalProp2": {},
               "additionalProp3": {}
              }
             }
            ]
           }
          ],
          "applicationSelections": [
           {
            "settings": [],
            "selectionId": "",
            "applicationId": "",
            "selected": true | false,
            "scenarioNames": [],
             "scenarioSelectionInfo": [
             {
            "scenarioName": "",
            "selected": true | false
             }
            ]
           }
          ]
         }
        ],
        "generateReport": true | false,
```

	"templateId": "" } **Note:** The arguments in bold font are mandatory.

Response

Returns command execution status and creates a test execution draft.

Table 25: Status codes

Status code	Description
200	Test execution draft created successfully.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Update the test execution draft

This command updates the test execution draft.

PUT `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute/draft`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Table continued…

Body	{
	"executionName": "",
	"linkId": "",
	"tags": [""],
	"description": "",
	"acquisitionMode":"LIVE|RECORDED",
	"waveformFolderPath":string
	"executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
	"testBenchId": "",
	"applicationRequests": [
	{
	"technology": "",
	"testCategory": {
	"type": "",
	"subType": ""
	},
	"loop": {
	"sources": [
	[
	{
	"name": "",
	"type": "",
	"signals": [
	{
	"name": "",
	"category": "TRANSMITTER | RECEIVER",
	"probeMethod": "SINGLE_ENDED | DIFFERENTIAL",
	"singleEnded": [],
	"differential": []
	}
	]
	}
	]
	],
	"applicationId": ""
	},
	"tag": "",
	"settings": [
	{
	"name": "",
	"type": "",
	"displayName": "",
	"group": "",
	"reference": [],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "",
	"unit": "",
	"description": "",
	"additionalProperties": {},

```
                "referenceGroupBy": {},
                "scenarioName": "",
                "stepName": "",
                "category": {
                 "type": "",
                 "subType": ""
                },
                "tag": "",
                "global": true | false,
                "mandatory": true | false,
                "editable": true | false,
                "deprecated": true | false,
                "rules": [
                 {
                  "name": "",
                  "type": "FILTER | VALIDATION",
                  "inputs": {
                   "additionalProp1": {},
                   "additionalProp2": {},
                   "additionalProp3": {}
                  },
                  "output": {
                   "additionalProp1": {},
                   "additionalProp2": {},
                   "additionalProp3": {}
                  }
                 }
                ]
               }
              ],
              "applicationSelections": [
               {
                "settings": [],
                "selectionId": "",
                "applicationId": "",
                "selected": true | false,
                "scenarioNames": [],
                 "scenarioSelectionInfo": [
                 {
                "scenarioName": "",
                "selected": true | false
                 }
                ]
               }
              ]
             }
            ],
            "generateReport": true | false,
            "templateId": ""
           }
```

	Note: The arguments in bold font are mandatory.

Response

Returns command execution status and updates the test execution draft.

Table 26: Status codes

Status code	Description
200	Test execution draft updated successfully.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete the test execution draft by id

This command deletes the given test execution draft by id.

DELETE `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute/draft/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution draft id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test execution draft.

Table 27: Status codes

Status code	Description
204	Test execution draft deleted successfully.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Merge test execution draft

This command merges the test execution draft.

```
POST https://<<host ip>>:<<portid>>/clarius/application/$execute/$merge/draft
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	{
	"executionId": "string",
	"executionName": "string",
	"linkId": "string",
	"tags": [
	"string"
	],
	"description": "string",
	"executionMode": "COMPLIANCE",
	"testBenchId": "string",
	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"applicationRequests": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},

```
    "loop": {
      "sources": [
        [
          {
            "name": "string",
            "type": "string",
            "signals": [
              {
                "name": "string",
                "category": "TRANSMITTER",
                "probeMethod": "SINGLE_ENDED",
                "singleEnded": [
                  {
                    "name": "string",
                    "label": "string",
                    "type": "string",
                    "polarity": "string",
                    "channel": "string",
                    "instrument": "string"
                  }
                ],
                "differential": [
                  {
                    "name": "string",
                    "label": "string",
                    "type": "string",
                    "polarity": "string",
                    "channel": "string",
                    "instrument": "string"
                  }
                ]
              }
            ]
          }
        ]
      ],
      "applicationId": "string"
    },
    "tag": "string",
    "settings": [
      {
        "name": "string",
        "type": "string",
        "displayName": "string",
        "group": "string",
        "reference": [
          "#/setting['temperature'] - To refer to application level setting named temperature ",
          "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
          "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup'
&& @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup
```

```
step of SignalTest scenario "
        ],
        "referenceType": "DEFAULT",
        "value": {},
        "constraints": "value != null AND value == true AND \"P05\" IN value",
        "unit": "string",
        "description": "string",
        "additionalProperties": {
          "additionalProp1": {},
          "additionalProp2": {},
          "additionalProp3": {}
        },
        "referenceGroupBy": {
          "name": "string",
          "value": "string"
        },
        "scenarioName": "string",
        "stepName": "string",
        "category": {
          "type": "display",
          "subType": "horizontal"
        },
        "tag": "string",
        "global": false,
        "mandatory": false,
        "editable": true,
        "deprecated": false,
        "rules": [
          {
            "name": "string",
            "type": "FILTER",
            "inputs": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            },
            "output": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            }
          }
        ]
      }
    ],
    "applicationSelections": [
      {
        "settings": [
          {
            "name": "string",
            "type": "string",
```

```
            "displayName": "string",
            "group": "string",
            "reference": [
              "#/setting['temperature'] - To refer to application level setting named temperature
",
              "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
              "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName ==
'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature
from ConnectSetup step of SignalTest scenario "
            ],
            "referenceType": "DEFAULT",
            "value": {},
            "constraints": "value != null AND value == true AND \"P05\" IN value",
            "unit": "string",
            "description": "string",
            "additionalProperties": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            },
            "referenceGroupBy": {
              "name": "string",
              "value": "string"
            },
            "scenarioName": "string",
            "stepName": "string",
            "category": {
              "type": "display",
              "subType": "horizontal"
            },
            "tag": "string",
            "global": false,
            "mandatory": false,
            "editable": true,
            "deprecated": false,
            "rules": [
              {
                "name": "string",
                "type": "FILTER",
                "inputs": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                },
                "output": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                }
              }
```

```
            ]
          }
        ],
        "selectionId": "string",
        "applicationId": "string",
        "selected": true,
        "scenarioNames": [
          "string"
        ],
        "scenarioSelectionInfo": [
          {
            "scenarioName": "string",
            "selected": true
          }
        ]
      }
    ]
  }
],
"generateReport": true,
"templateId": "string"
}
```


Note: The arguments in bold font are mandatory.

Response

Returns command execution status and merges the test execution draft.

Table 28: Status codes

Status code	Description
200	Test execution draft merged successfully.
400	The validation of the request failed. `{ "errors": [ { "errorCode": "string", "errorDescription": "string", "errorLocation": "string", "errorTrace": "string" } ] }`
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Table continued…

Status code	Description
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get all applications test execution status

This command retrieves the application test execution status.

```
GET https://<<host ip>>:<<portid>>/clarius/application/$execute/status?
page=<<page>>&pageSize=<<pageSize>>&from=<<from>>&to=<<to>>&applications=<<applicat
ions>>&scenarios=<<scenarios>>&statuses=<<statuses>>&tag=<<tag>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
page[2]	Specify the page value Default value is : 1
pageSize[2]	Specify the page size Default value is : 10
from[2]	Specify the "from" date (YYYY-MM-DD)
to[2]	Specify the "to" date (YYYY-MM-DD)
applications[2]	Array of application ids
scenarios[2]	Array of scenarios names
statuses[2]	Array of statuses
tag[2]	Tag value

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command and application test execution status in JSON format.

[2] This parameter is optional.

Table 29: Status codes

Status code	Description

200	Test execution status retrieved successfully.
	
	[
	{
	"executionId": "string",
	"executionName": "string",
	"acquisitionMode":"LIVE|RECORDED",
	"waveformFolderPath":string
	"linkId": "string",
	"tags": [
	"string"
	],
	"applicationRequests": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},
	"loop": {
	"sources": [
	[
	{
	"name": "string",
	"type": "string",
	"signals": [
	{
	"name": "string",
	"category": "TRANSMITTER",
	"probeMethod": "SINGLE_ENDED",
	"singleEnded": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	],
	"differential": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"

Table continued…

Status code	Description

	}
	]
	}
	]
	}
	]
	],
	"applicationId": "string"
	},
	"tag": "string",
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [
	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",
	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"referenceGroupBy": {
	"name": "string",
	"value": "string"
	},
	"scenarioName": "string",
	"stepName": "string",
	"category": {
	"type": "display",
	"subType": "horizontal"
	},
	"tag": "string",
	"global": false,
	"mandatory": false,
	"editable": true,

Table continued…

Status code	Description

	"deprecated": false,
	"rules": [
	{
	"name": "string",
	"type": "FILTER",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"applicationSelections": [
	{
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [
	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",
	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"referenceGroupBy": {
	"name": "string",
	"value": "string"

Table continued…

Status code	Description
	}, "scenarioName": "string", "stepName": "string", "category": { "type": "display", "subType": "horizontal" }, "tag": "string", "global": false, "mandatory": false, "editable": true, "deprecated": false, "rules": [{ "name": "string", "type": "FILTER", "inputs": { "additionalProp1": {}, "additionalProp2": {}, "additionalProp3": {} }, "output": { "additionalProp1": {}, "additionalProp2": {}, "additionalProp3": {} } }] }], "selectionId": "string", "applicationId": "string", "selected": true, "scenarioNames": ["string"], "scenarioSelectionInfo": [{ "scenarioName": "string", "selected": true }] }] }], "description": "string", "executionMode": "COMPLIANCE | USER | CHARACTERIZATION", "testBenchId": "string",

Table continued…

Status code	Description

	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"status": "PASSED | FAILED",
	"totalScenarios": 0,
	"createdTime": "YYYY/MM/DD",
	"startTime": "YYYY/MM/DD",
	"endTime": "YYYY/MM/DD",
	"executedScenarios": 0,
	"testbenchDetails": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"softwareInfo": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}
	]

400	The validation of the request failed.
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get the total count of test execution data

This command retrieves the total count of test execution data that can be used for displaying the execution status list.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/application/$execute/status/pagination`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test execution count in JSON format.

Table 30: Status codes

Status code	Description

200	Test execution data count retrieved successfully.
	
	{
	"totalSize": ""
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test execution status by id

This command retrieves the test execution status of an application for the given execution id.

GET https://**<<host ip>>:<<portid>>**/clarius/application/$execute/status/**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns text execution status of all applications in JSON format.

Table 31: Status codes

Status code	Description

200	Test execution status retrieved successfully.
	
	{
	"executionId": "string",
	"executionName": "string",
	"acquisitionMode": "LIVE|RECORDED",
	"waveformFolderPath":string
	"linkId": "string",
	"tags": [
	"string"
	],
	"description": "string",
	"testStatuses": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},
	"loop": {
	"sources": [
	[
	{
	"name": "string",
	"type": "string",
	"signals": [
	{
	"name": "string",
	"category": "TRANSMITTER",
	"probeMethod": "SINGLE_ENDED",
	"singleEnded": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	],
	"differential": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"

Table continued…

Status code	Description

	}
	]
	}
	]
	}
	]
	],
	"applicationId": "string"
	},
	"tag": "string",
	"applicationStatuses": [
	{
	"selectionId": "string",
	"applicationId": "string",
	"applicationName": "string",
	"applicationDescription": "string",
	"selected": true,
	"scenarioStatuses": [
	{
	"name": "string",
	"id": "string",
	"status": "PASSED",
	"statusDescription": "string",
	"startTime": "2024-11-12",
	"endTime": "2024-11-12",
	"sourceName": "string",
	"iterationNumber": 0,
	"errors": {
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	},
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"selected": true,
	"stepStatuses": [
	{
	"name": "string",
	"id": "string",
	"status": "PASSED",
	"statusDescription": "string",

Table continued…

Status code	Description

	"startTime": "2024-11-12",
	"endTime": "2024-11-12",
	"sourceName": "string",
	"iterationNumber": 0,
	"errors": {
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	},
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	]
	}
	]
	}
	],
	"executionMode": "COMPLIANCE | USER | CHARACTERIZATION",
	"testBenchId": "string",
	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"totalScenarios": 0,
	"status": "PASSED | FAILED",
	"executedScenarios": 0,
	"startTime": "YYYY/MM/DD",
	"endTime": "YYYY/MM/DD"
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Download test data for the given test execution id

This command downloads the test data for the given test execution id.

```
GET https://<<host ip>>:<<portid>>/clarius/test/<<id>>/$download?
internal=<<internal>>&format=<<format>>&incResults=<<incResults>>&incSettings=<<inc
Settings>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution id
internal[3]	Flag to display internal result data. Either "true" or "false"
format[3]	Test data format Default format is **csv**
incResults[3]	Either "true" or "false"
incSettings[3]	Either "true" or "false"

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and downloads the complete test data as a zip file.

Table 32: Status codes

Status code	Description
200	Test data downloaded successfully.
400	Validation of query parameters failed.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test execution result

This command retrieves the test execution result for the given test execution id.

```
GET https://<<host ip>>:<<portid>>/clarius/application/$execute/results/<<id>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test execution id

[3] This parameter is optional.

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test results in JSON format.

Table 33: Status codes

Status code	Description

200	Test execution results were retrieved successfully.
	
	{
	"executionId": "",
	"executionName": "",
	"acquisitionMode": "LIVE|RECORDED",
	"waveformFolderPath":string
	"linkId": "",
	"tags": [""],
	"applicationRequests": [
	{
	"technology": "",
	"testCategory": {
	"type": "",
	"subType": ""
	},
	"loop": {
	"sources": [],
	"applicationId": ""
	},
	"tag": "",
	"settings": [
	{
	"name": "",
	"type": "",
	"displayName": "",
	"group": "",
	"reference": [],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "",
	"unit": "",
	"description": "",
	"additionalProperties": {},
	"referenceGroupBy": {},
	"scenarioName": "",
	"stepName": "",
	"category": {},
	"tag": "",

Table continued…

Status code	Description
	"global": true | false, "mandatory": true | false, "editable": true | false, "deprecated": true | false, "rules": [] }], "applicationSelections": [] }], "description": "", "executionMode": "COMPLIANCE | USER | CHARACTERIZATION", "testBenchId": "", "status": "PASSED | FAILED", "totalScenarios": 0, "createdTime": "YYYY/MM/DD-HH:MM:SS", "startTime": "YYYY/MM/DD-HH:MM:SS", "endTime": "YYYY/MM/DD-HH:MM:SS", "executedScenarios": 0, "testbenchDetails": {}, "softwareInfo": {} }
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test waveform id

This command fetches the list of waveform ids related to the test execution id, application id, scenario name, or step name.

```
GET https://<<host ip>>:<<portid>>/clarius/application/$execute/results/<<executionId>>/waveform?applicationId=<<applicationId>>&scenarioName=<<scenarioName>>&stepName=<<stepName>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
executionId	Test execution Id
applicationId[4]	Application Id
scenarioName[4]	Test scenario name
stepName[4]	Step name

[4] This parameter is optional.

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and waveform id lists in JSON format.

Table 34: Status codes

Status code	Description
200	Waveform id list retrieved successfully. `[ "<<waveform id>>.wfm", "<<waveform id>.wfm" ]`

400	The validation of query parameters failed.
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string"
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Download test waveforms

This command downloads the test waveforms as a zip file using waveform ids.

POST `https://<<host ip>>:<<portid>>/clarius/waveform/$download`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Table continued…

Body	["<<waveform id>>.wfm", "<<waveform id>>.wfm"]

The list of waveform ids can also be retrieved using the API defined in the *Get test waveform id* on page 71.

Response

Returns command execution status and downloads test waveforms.

Table 35: Status codes

Status code	Description
200	Test waveforms were downloaded successfully in (.zip) format.
400	The validation of the request failed.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.

Delete waveforms of executed test

This command deletes the waveforms of the executed tests, specified by ID.

```
DELETE https://<<host ip>>:<<portid>>/clarius/application/$execute/results/waveform
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	["executionId1", "executionId2", "executionId3"]

Response

Returns command execution status.

Table 36: Status codes

Status code	Description
204	Waveforms deleted successfully
400	Waveforms delete failed
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of notifications

This command retrieves the list of notifications.

```
GET https://<<host ip>>:<<portid>>/clarius/notification/$fetch?
notificationId=<<notificationId>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
notificationId	Notification id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and list of notifications in JSON format.

Table 37: Status codes

Status code	Description

200	List of notifications retrieved successfully.
	
	[
	{
	"id": "",
	"executionId": "",
	"selectionId": "",
	"applicationId": "",
	"scenarioReferenceId": "",
	"stepReferenceId": "",
	"applicationName": "",
	"scenarioName": "",
	"stepName": "",
	"status": ["SENT | NOTIFIED | ACKNOWLEDGED"],
	"message": "",
	"possibleActions": ["STOP | SKIP | PAUSE | RESUME"],
	"performedAction": ["STOP | SKIP | PAUSE | RESUME"],
	"notificationTime": "YYYY/MM/DD-HH:MM:SS"
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Application intervention

This command intervenes in an application execution.

POST `https://<<host ip>>:<<portid>>/clarius/application/$execute/intervene`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Table continued…

Body	<pre>{
	"notificationId": "",
	"executionId": "",
	"selectionId": "",
	"applicationId": "",
	"activityId": "",
	"taskId": "",
	"action": "STOP | SKIP | PAUSE | RESUME"
	}</pre>

Response

Returns command execution status.

Table 38: Status codes

Status code	Description
200	Application intervention requests were processed successfully

400	The validation of the request failed.
	<pre>{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}</pre>

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of unacknowledged notifications based on query parameters

This command retrieves the list of all unacknowledged notifications based on the query parameters.

```
GET https://<<host ip>>:<<portid>>/clarius/notification?
executionId=<<executionId>>&applicationId=<<applicationId>>&scenarioName=<<scenario
Name>>&stepName=<<stepName>>
```

URL Parameters

Name	Description
host ip	Host ip address

Table continued…

Name	Description
portid	Port id
executionId	Test execution id
applicationId	Application id
scenarioName	Test scenario name
stepName	Step name

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and notification list in JSON format.

Table 39: Status codes

Status code	Description

200	List of notifications retrieved successfully.
	
	[
	{
	"id": "",
	"executionId": "",
	"selectionId": "",
	"applicationId": "",
	"scenarioReferenceId": "",
	"stepReferenceId": "",
	"applicationName": "",
	"scenarioName": "",
	"stepName": "",
	"status": ["SENT | NOTIFIED | ACKNOWLEDGED"],
	"message": "",
	"possibleActions": ["STOP | SKIP | PAUSE | RESUME"],
	"performedAction": ["STOP | SKIP | PAUSE | RESUME"],
	"notificationTime": "YYYY/MM/DD-HH:MM:SS"
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Test sequence

The test sequence API contains a set of rest endpoints that are used for creating, retrieving, updating, and deleting the test sequences.

Create a new sequence

This command creates a new sequence.

POST `https://<<host ip>>:<<portid>>/clarius/sequence`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	{
	"executionId": "string",
	"executionName": "string",
	"acquisitionMode":LIVE | RECORDED,
	"waveformFolderPath":string
	"linkId": "string",
	"tags": [
	"string"
	],
	"description": "string",
	"executionMode": "COMPLIANCE",
	"testBenchId": "string",
	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"applicationRequests": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},
	"loop": {
	"sources": [
	[
	{
	"name": "string",
	"type": "string",
	"signals": [
	{
	"name": "string",
	"category": "TRANSMITTER",
	"probeMethod": "SINGLE_ENDED",
	"singleEnded": [
	{
	"name": "string",

```
                "label": "string",
                "type": "string",
                "polarity": "string",
                "channel": "string",
                "instrument": "string"
              }
            ],
            "differential": [
              {
                "name": "string",
                "label": "string",
                "type": "string",
                "polarity": "string",
                "channel": "string",
                "instrument": "string"
              }
            ]
          }
        ]
      }
    ]
  ],
  "applicationId": "string"
},
"tag": "string",
"settings": [
  {
    "name": "string",
    "type": "string",
    "displayName": "string",
    "group": "string",
    "reference": [
      "#/setting['temperature'] - To refer to application level setting named temperature ",
      "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
      "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup'
&& @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup
step of SignalTest scenario "
    ],
    "referenceType": "DEFAULT",
    "value": {},
    "constraints": "value != null AND value == true AND \"P05\" IN value",
    "unit": "string",
    "description": "string",
    "additionalProperties": {
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    },
    "referenceGroupBy": {
```

```
            "name": "string",
            "value": "string"
          },
          "scenarioName": "string",
          "stepName": "string",
          "category": {
            "type": "display",
            "subType": "horizontal"
          },
          "tag": "string",
          "global": false,
          "mandatory": false,
          "editable": true,
          "deprecated": false,
          "rules": [
            {
              "name": "string",
              "type": "FILTER",
              "inputs": {
                "additionalProp1": {},
                "additionalProp2": {},
                "additionalProp3": {}
              },
              "output": {
                "additionalProp1": {},
                "additionalProp2": {},
                "additionalProp3": {}
              }
            }
          ]
        }
      ],
      "applicationSelections": [
        {
          "settings": [
            {
              "name": "string",
              "type": "string",
              "displayName": "string",
              "group": "string",
              "reference": [
                "#/setting['temperature'] - To refer to application level setting named temperature
",
                "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
                "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName ==
'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature
from ConnectSetup step of SignalTest scenario "
              ],
              "referenceType": "DEFAULT",
```

```
      "value": {},
      "constraints": "value != null AND value == true AND \"P05\" IN value",
      "unit": "string",
      "description": "string",
      "additionalProperties": {
        "additionalProp1": {},
        "additionalProp2": {},
        "additionalProp3": {}
      },
      "referenceGroupBy": {
        "name": "string",
        "value": "string"
      },
      "scenarioName": "string",
      "stepName": "string",
      "category": {
        "type": "display",
        "subType": "horizontal"
      },
      "tag": "string",
      "global": false,
      "mandatory": false,
      "editable": true,
      "deprecated": false,
      "rules": [
        {
          "name": "string",
          "type": "FILTER",
          "inputs": {
            "additionalProp1": {},
            "additionalProp2": {},
            "additionalProp3": {}
          },
          "output": {
            "additionalProp1": {},
            "additionalProp2": {},
            "additionalProp3": {}
          }
        }
      ]
    }
  ],
  "selectionId": "string",
  "applicationId": "string",
  "selected": true,
  "scenarioNames": [
    "string"
  ],
  "scenarioSelectionInfo": [
    {
```

```
                "scenarioName": "string",
                "selected": true
              }
            ]
          }
        ]
      }
    ],
    "generateReport": true,
    "templateId": "string"
  }
```


Note: The arguments in bold font are mandatory.

Response

Returns command execution status and creates a new sequence.

Table 40: Status codes

Status code	Description
200	Test sequence created successfully.
400	The validation of the request failed. `{ "errors": [ { "errorCode": "string", "errorDescription": "string", "errorLocation": "string", "errorTrace": "string" } ] }`
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of all sequences

This command retrieves the list of all sequences.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/sequence`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and list of test sequences in JSON format.

Table 41: Status codes

Status code	Description

200	List of test sequences retrieved successfully.
	
	[
	{
	"executionId": "string",
	"executionName": "string",
	"acquisitionMode":LIVE | RECORDED,
	"waveformFolderPath":string
	"linkId": "string",
	"tags": [
	"string"
	],
	"description": "string",
	"executionMode": "COMPLIANCE",
	"testBenchId": "string",
	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"applicationRequests": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},
	"loop": {
	"sources": [
	[
	{
	"name": "string",
	"type": "string",
	"signals": [
	{

Table continued…

Status code	Description

	"name": "string",
	"category": "TRANSMITTER",
	"probeMethod": "SINGLE_ENDED",
	"singleEnded": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	],
	"differential": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	]
	}
	]
	}
	]
	],
	"applicationId": "string"
	},
	"tag": "string",
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [
	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",

Table continued…

Status code	Description

	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"referenceGroupBy": {
	"name": "string",
	"value": "string"
	},
	"scenarioName": "string",
	"stepName": "string",
	"category": {
	"type": "display",
	"subType": "horizontal"
	},
	"tag": "string",
	"global": false,
	"mandatory": false,
	"editable": true,
	"deprecated": false,
	"rules": [
	{
	"name": "string",
	"type": "FILTER",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"applicationSelections": [
	{
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [

Table continued…

Status code	Description

	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",
	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"referenceGroupBy": {
	"name": "string",
	"value": "string"
	},
	"scenarioName": "string",
	"stepName": "string",
	"category": {
	"type": "display",
	"subType": "horizontal"
	},
	"tag": "string",
	"global": false,
	"mandatory": false,
	"editable": true,
	"deprecated": false,
	"rules": [
	{
	"name": "string",
	"type": "FILTER",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]

Table continued…

Status code	Description

	}
	],
	"selectionId": "string",
	"applicationId": "string",
	"selected": true,
	"scenarioNames": [
	"string"
	],
	"scenarioSelectionInfo": [
	{
	"scenarioName": "string",
	"selected": true
	}
	]
	}
	]
	}
	],
	"generateReport": true,
	"templateId": "string"
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Update the test sequence

This command updates the test sequence.

PUT `https://`**<<host ip>>:<<portid>>**`/clarius/sequence`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	{ "executionId": "string", **"executionName"**: "string", "acquisitionMode":LIVE|RECORDED, "waveformFolderPath":string

```
"linkId": "string",
"tags": [
  "string"
],
"description": "string",
"executionMode": "COMPLIANCE",
"testBenchId": "string",
"acquisitionMode": "LIVE",
"waveformFolderPath": "string",
"applicationRequests": [
  {
    "technology": "string",
    "testCategory": {
      "type": "TX",
      "subType": "TEST"
    },
    "loop": {
      "sources": [
        [
          {
            "name": "string",
            "type": "string",
            "signals": [
              {
                "name": "string",
                "category": "TRANSMITTER",
                "probeMethod": "SINGLE_ENDED",
                "singleEnded": [
                  {
                    "name": "string",
                    "label": "string",
                    "type": "string",
                    "polarity": "string",
                    "channel": "string",
                    "instrument": "string"
                  }
                ],
                "differential": [
                  {
                    "name": "string",
                    "label": "string",
                    "type": "string",
                    "polarity": "string",
                    "channel": "string",
                    "instrument": "string"
                  }
                ]
              }
            ]
          }
        ]
```

```
      ],
      "applicationId": "string"
    },
    "tag": "string",
    "settings": [
      {
        "name": "string",
        "type": "string",
        "displayName": "string",
        "group": "string",
        "reference": [
          "#/setting['temperature'] - To refer to application level setting named temperature ",
          "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
          "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup'
&& @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup
step of SignalTest scenario "
        ],
        "referenceType": "DEFAULT",
        "value": {},
        "constraints": "value != null AND value == true AND \"P05\" IN value",
        "unit": "string",
        "description": "string",
        "additionalProperties": {
          "additionalProp1": {},
          "additionalProp2": {},
          "additionalProp3": {}
        },
        "referenceGroupBy": {
          "name": "string",
          "value": "string"
        },
        "scenarioName": "string",
        "stepName": "string",
        "category": {
          "type": "display",
          "subType": "horizontal"
        },
        "tag": "string",
        "global": false,
        "mandatory": false,
        "editable": true,
        "deprecated": false,
        "rules": [
          {
            "name": "string",
            "type": "FILTER",
            "inputs": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
```

```
            },
            "output": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            }
          }
        ]
      }
    ],
    "applicationSelections": [
      {
        "settings": [
          {
            "name": "string",
            "type": "string",
            "displayName": "string",
            "group": "string",
            "reference": [
              "#/setting['temperature'] - To refer to application level setting named temperature
",
              "#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To
refer to setting named temperature from SignalTest scenario ",
              "#/setting[?(@.scenarioName == 'SignalTest' && @.stepName ==
'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature
from ConnectSetup step of SignalTest scenario "
            ],
            "referenceType": "DEFAULT",
            "value": {},
            "constraints": "value != null AND value == true AND \"P05\" IN value",
            "unit": "string",
            "description": "string",
            "additionalProperties": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            },
            "referenceGroupBy": {
              "name": "string",
              "value": "string"
            },
            "scenarioName": "string",
            "stepName": "string",
            "category": {
              "type": "display",
              "subType": "horizontal"
            },
            "tag": "string",
            "global": false,
            "mandatory": false,
            "editable": true,
```

```
            "deprecated": false,
            "rules": [
              {
                "name": "string",
                "type": "FILTER",
                "inputs": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                },
                "output": {
                  "additionalProp1": {},
                  "additionalProp2": {},
                  "additionalProp3": {}
                }
              }
            ]
          }
        ],
        "selectionId": "string",
        "applicationId": "string",
        "selected": true,
        "scenarioNames": [
          "string"
        ],
        "scenarioSelectionInfo": [
          {
            "scenarioName": "string",
            "selected": true
          }
        ]
      }
    ]
  }
],
"generateReport": true,
"templateId": "string"
}
```


Note: The arguments in bold font are mandatory.

Response

Returns command execution status and updates the test sequence.

Table 42: Status codes

Status code	Description
200	Test sequence updated successfully.

Table continued…

Status code	Description

400	The validation of request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get the sequence by id

This command retrieves the test sequence details by its id.

GET `https://`**<<host ip>>:<<portid>>**`/clarius/sequence/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test sequence id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test sequence details in JSON format.

Table 43: Status codes

Status code	Description

200	Test sequence retrieved successfully.
	
	{
	"executionId": "string",
	"executionName": "string",
	"acquisitionMode":LIVE|RECORDED,
	"waveformFolderPath":string
	"linkId": "string",
	"tags": [
	"string"
	],
	"description": "string",
	"executionMode": "COMPLIANCE",
	"testBenchId": "string",
	"acquisitionMode": "LIVE",
	"waveformFolderPath": "string",
	"applicationRequests": [
	{
	"technology": "string",
	"testCategory": {
	"type": "TX",
	"subType": "TEST"
	},
	"loop": {
	"sources": [
	[
	{
	"name": "string",
	"type": "string",
	"signals": [
	{
	"name": "string",
	"category": "TRANSMITTER",
	"probeMethod": "SINGLE_ENDED",
	"singleEnded": [
	{
	"name": "string",
	"label": "string",
	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	],
	"differential": [
	{
	"name": "string",
	"label": "string",

Table continued…

Status code	Description

	"type": "string",
	"polarity": "string",
	"channel": "string",
	"instrument": "string"
	}
	]
	}
	]
	}
	]
	],
	"applicationId": "string"
	},
	"tag": "string",
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [
	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",
	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"referenceGroupBy": {
	"name": "string",
	"value": "string"
	},
	"scenarioName": "string",
	"stepName": "string",
	"category": {
	"type": "display",
	"subType": "horizontal"
	},
	"tag": "string",

Table continued…

Status code	Description

	"global": false,
	"mandatory": false,
	"editable": true,
	"deprecated": false,
	"rules": [
	{
	"name": "string",
	"type": "FILTER",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"applicationSelections": [
	{
	"settings": [
	{
	"name": "string",
	"type": "string",
	"displayName": "string",
	"group": "string",
	"reference": [
	"#/setting['temperature'] - To refer to application level setting named temperature ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.name == 'temperature')]- To refer to setting named temperature from SignalTest scenario ",
	"#/setting[?(@.scenarioName == 'SignalTest' && @.stepName == 'ConnectSetup' && @.name == 'temperature')] - To refer to setting named temperature from ConnectSetup step of SignalTest scenario "
	],
	"referenceType": "DEFAULT",
	"value": {},
	"constraints": "value != null AND value == true AND \"P05\" IN value",
	"unit": "string",
	"description": "string",
	"additionalProperties": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},

Table continued…

Status code	Description

	"referenceGroupBy": {
	"name": "string",
	"value": "string"
	},
	"scenarioName": "string",
	"stepName": "string",
	"category": {
	"type": "display",
	"subType": "horizontal"
	},
	"tag": "string",
	"global": false,
	"mandatory": false,
	"editable": true,
	"deprecated": false,
	"rules": [
	{
	"name": "string",
	"type": "FILTER",
	"inputs": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	},
	"output": {
	"additionalProp1": {},
	"additionalProp2": {},
	"additionalProp3": {}
	}
	}
	]
	}
	],
	"selectionId": "string",
	"applicationId": "string",
	"selected": true,
	"scenarioNames": [
	"string"
	],
	"scenarioSelectionInfo": [
	{
	"scenarioName": "string",
	"selected": true
	}
	]
	}
	]
	}
	],

Table continued…

Status code	Description
	"generateReport": true, "templateId": "string" }
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete the test sequence by id

This command deletes the test sequence by id.

DELETE `https://`**<<host ip>>:<<portid>>**`/clarius/sequence/`**<<id>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
id	Test sequence id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test sequence.

Table 44: Status codes

Status code	Description
200	Test sequence deleted successfully.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Test logs

The logs API contains the set of rest endpoints that are used for retrieving and deleting the logs from the database.

Get test logs

This command retrieves the test logs.

```
GET https://<<host ip>>:<<portid>>/clarius/log?
from=<<from>>&to=<<to>>&page=<<page>>&pageSize=<<pageSize>>&filterList=<<filterList
>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
from	From date (YYYY-MM-DD HH:MM)
to	To date (YYYY-MM-DD HH:MM)
page	Page
pageSize	Page size
filterList	Filter the logs based on the objects such as: • level • origin • host • service • component • transactionId • transactionType

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test logs in JSON format.

Table 45: Status codes

Status code	Description

200	List of log messages retrieved successfully.
	[
	{
	"id": "",
	"level": "",
	"logTime": "YYYY/MM/DD-HH:MM:SS",
	"origin": "",
	"host": "",
	"service": "",
	"component": "",
	"transactionId": "",
	"transactionType": "",
	"additionalTags": {
	"additionalProp1": "",
	"additionalProp2": "",
	"additionalProp3": ""
	},
	"additionalTagsJSONString": "",
	"user": "",
	"source": "",
	"message": "",
	"trace": ""
	}
	]

400	The validation of the request failed.
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete test logs

This command deletes the test logs.

```
DELETE https://<<host ip>>:<<portid>>/clarius/log?
from=<<from>>&to=<<to>>&filterList=<<filterList>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
from	From date (YYYY-MM-DD HH:MM)
to	To date (YYYY-MM-DD HH:MM)
filterList	Filter list

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test logs.

Table 46: Status codes

Status code	Description
204	Test logs were deleted successfully.

400	The validation of query parameters failed.
	
	{
	"errorCode": "",
	"errorDescription": "",
	"errorLocation": ""
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Download test logs

This command downloads the test logs as a (.txt) file.

```
DELETE https://<<host ip>>:<<portid>>/clarius/log/$download?
from=<<from>>&to=<<to>>&page=<<page>>&pageSize=<<pageSize>>&offset=<<offset>>&filte
rList=<<filterList>>
```

URL Parameters

Name	Description
host ip	Host ip address

Table continued…

Name	Description
portid	Port id
from	From date (YYYY-MM-DD HH:MM)
to	To date (YYYY-MM-DD HH:MM)
page	Page
pageSize	Page size
offset	Offset
filterList	Filter list

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and downloads the test logs as a (.txt) file

Table 47: Status codes

Status code	Description
200	Test logs were downloaded successfully as a (.txt) file.

400	The validation of the request failed.
	
	{
	"errors": [
	{
	"errorCode": "string",
	"errorDescription": "string",
	"errorLocation": "string",
	"errorTrace": "string"
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get the total count of test logs

This command retrieves the total count of test logs.

```
GET https://<<host ip>>:<<portid>>/clarius/log/pagination?
from=<<from>>&to=<<to>>&filterList=<<filterList>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
from	From date (YYYY-MM-DD HH:MM)
to	To date (YYYY-MM-DD HH:MM)
filterList	Filter list

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and total count of test logs in JSON format.

Table 48: Status codes

Status code	Description

200	Logs count retrieved successfully.
	
	{
	"documentSize": ""
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get distinct value of test log message

This command retrieves the distinct values of log message fields.

```
GET https://<<host ip>>:<<portid>>/clarius/log/values?fieldNames=<<fieldNames>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Table continued…

Name	Description
fieldNames	The list of field names are: • level • origin • host • service • component • transactionId • transactionType

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and test log message in JSON format.

Table 49: Status codes

Status code	Description

200	Distinct values of log message fields were retrieved successfully.
	
	{
	"field1": [
	"value1",
	"value2"
	],
	"field2": [
	"value3",
	"value4"
	]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Reports

The reports API contains the set of rest endpoints that are used for generating, searching, retrieving, and deleting the reports from the database using predefined templates. It also has an additional endpoint to list down all the reports present in the database based on the query parameter such as **executionId** and **templateId**.

Generate test report

This command generates a test report.

POST `https://<<host ip>>:<<portid>>/clarius/report/generate`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	<pre>{
	"**executionId**" : "",
	"**templateId**" : "",
	"**reportName**" : "",
	"customLogoData": "",
	"userComment":""
	}</pre> **Note:** The arguments in bold font are mandatory.

Response

Returns command execution status and generates a test report.

Table 50: Status codes

Status code	Description

200	Test reports were generated successfully.
	<pre>{
	"executionId": "",
	"executionName": "",
	"reportName": "",
	"userComment": "",
	"testStartTime": "YYYY-MM-DD",
	"testEndTime": "YYYY-MM-DD",
	"generatedDate": "YYYY-MM-DD",
	"reportId": "",
	"templateId": "",
	"status": "PASS | FAIL | RUNNING | SUBMITTED",
	"statusDescription": "",
	"applicationId": [""]
	}</pre>

Table continued…

Status code	Description
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
409	Conflict.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of test reports based on query parameter

This command retrieves the list of reports generated based on the query parameters.

```
GET https://<<host ip>>:<<port id>>/clarius/report?
executionId=<<execution-id>>&executionName=<<execution-name>>&fromDate=<<YYYY-MM-
DD>>&toDate=<<YYYY-MM-DD>>&reportName=<<report-name>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
executionId	Test execution id
executionName	Test execution name
fromDate	Select the from date (YYYY-MM-DD) to filter the report generated by timestamp.
toDate	Select the to date (YYYY-MM-DD) to filter the report generated by timestamp. toDate must be greater than the fromDate.
reportName	Report name

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and reports list in JSON format.

Table 51: Status codes

Status code	Description

200	Reports list retrieved successfully.
	
	{
	"executionId": "",
	"executionName": "",
	"reportName": "",
	"userComment": "",
	"testStartTime": "YYYY/MM/DD-HH:MM:SS",
	"testEndTime": "YYYY/MM/DD-HH:MM:SS",
	"generatedDate": "YYYY/MM/DD-HH:MM:SS",
	"reportId": "",
	"templateId": "",
	"status": "PASS | FAIL | RUNNING | SUBMITTED",
	"statusDescription": "",
	"applicationId": [""]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Delete a report

This command deletes a report for the given test execution Id.

```
DELETE https://<<host ip>>:<<portid>>/clarius/report?executionId=<<execution-Id>>&templateId=<<template-Id>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
executionId	Test execution id
templateId	Report template id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the test report.

Table 52: Status codes

Status code	Description
204	Report deleted successfully.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Generate pdf report

This command generates the pdf report for the given test execution id.

GET https://**<<host ip>>:<<portid>>**/clarius/report/**<<executionId>>**/**<<templateId>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
executionId	Test execution id
templateId	Template id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and generates the test report in pdf format.

Table 53: Status codes

Status code	Description
200	Test reports were generated successfully in (.pdf) format.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached.
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

Get list of reports based on search parameter

This command retrieves the list of test reports generated based on the search parameters.

```
GET https://<<host ip>>:<<port id>>/clarius/report/search?
searchVariable=<<searchVariable>>
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
searchVariable	Search variable of the report.

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and reports list in JSON format.

Table 54: Status codes

Status code	Description

200	Reports list retrieved successfully.
	
	{
	"executionId": "",
	"executionName": "",
	"reportName": "",
	"userComment": "",
	"testStartTime": "YYYY/MM/DD-HH:MM:SS",
	"testEndTime": "YYYY/MM/DD-HH:MM:SS",
	"generatedDate": "YYYY/MM/DD-HH:MM:SS",
	"reportId": "",
	"templateId": "",
	"status": "PASS | FAIL | RUNNING | SUBMITTED",
	"statusDescription": "",
	"applicationId": [""]
	}

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	The endpoint cannot be reached
500	The system encountered an error and the user has to contact the administrator to resolve the problem.

User management

The user management API contains the set of rest endpoints that are used for creating, retrieving, updating, and deleting the users information from the database.

Add a new user (Admin only)

This command adds a new user account.

POST `https://<<host ip>>:<<portid>>/clarius/users`

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	<pre>{
	"username": "",
	"password": "",
	"name": "",
	"user_group": "usradmingrp | usrgrp",
	"emailId": "",
	"organization": "",
	"contact_no": ""
	}</pre>

Response

Returns command execution status and adds a new user.

Table 55: Status codes

Status code	Description

201	User added successfully.
	<pre>{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp",
	"Status": "LOCKED | UNLOCKED"
	}</pre>

400	Bad requests.

Table continued…

Status code	Description
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Unlock a user account (Admin only)

This command unlocks a user account.

POST https://**<<host ip>>:<<portid>>**/clarius/users/unlock/**<<username>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
username	user name of locked account

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and unlocks a user account.

Table 56: Status codes

Status code	Description
200	User account were unlocked successfully.
400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Get users information

This command retrieves the following.

For administrator: Retrieves information about all users from the database.

For non-administrator: Retrieves information about current user account from the database.

GET https://**<<host ip>>:<<portid>>**/clarius/users

URL Parameters

Name	Description
host ip	Host ip address

Table continued…

Name	Description
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and users information in JSON format.

Table 57: Status codes

Status code	Description

200	Users information retrieved successfully.
	
	[
	{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp",
	"Status": "LOCKED | UNLOCKED"
	}
	]

401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	Users not found.

Get self-profile information

This command retrieves self-profile information from the database.

```
GET https://<<host ip>>:<<portid>>/clarius/users/profile
```

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and self-profile information in JSON format.

Table 58: Status codes

Status code	Description

200	Self-profile information retrieved successfully.
	
	{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp",
	"Status": "LOCKED | UNLOCKED"
	}

400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	User not found.

Get specific user information

This command retrieves the following.

For administrator: Retrieves information about specific user from the database.

For non-administrator: Retrieves information about current user account from the database.

GET https://**<<host ip>>:<<portid>>**/clarius/users/**<<username>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
username	user name of an account

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and user information in JSON format.

Table 59: Status codes

Status code	Description

200	Users information retrieved successfully.
	
	{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp",
	"Status": "LOCKED | UNLOCKED"
	}

400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	User not found.

Update user password

This command updates the password of the current user.

PUT https://**<<host ip>>:<<portid>>**/clarius/users

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id

Request

Header	Authorization: {Bearer access token}

Body	{
	"old_password":"",
	"new_password":""
	}

Response

Returns command execution status.

Table 60: Status codes

Status code	Description

200	Password updated successfully.
	
	{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp"
	}

400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	User not found.

Update a user account (Admin only)

This command updates the details of a user account.

PUT `https://`**<<host ip>>:<<portid>>**`/clarius/users/`**<<username>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
username	user name of an account

Request

Header	Authorization: {Bearer access token}

Body	{
	"password": "",
	"name": "",
	"emailId": "",
	"organization": "",
	"contact_no": ""
	}

Response

Returns command execution status.

Table 61: Status codes

Status code	Description

200	User details were updated successfully.
	
	{
	"Name": "",
	"EmailId": "",
	"Organization": "",
	"ContactNumber": "",
	"Username": "",
	"UserGroup": "usradmingrp | usrgrp"
	}

400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.
404	User not found.

Delete user account (Admin only)

This command deletes the user account.

DELETE https://**<<host ip>>:<<portid>>**/clarius/users/**<<username>>**

URL Parameters

Name	Description
host ip	Host ip address
portid	Port id
username	user name of an account

Request

Header	Authorization: {Bearer access token}
Body	-

Response

Returns command execution status and deletes the user account.

Table 62: Status codes

Status code	Description
200	User accounts were deleted successfully.
400	Bad requests.
401	Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Table continued…

Status code	Description
404	User account not found.

Clarius SDK Automation

Introduction

Clarius Automation Framework provides a robust mechanism using optimized computing techniques to decrease the run time by executing the measurements parallely. It also provides test data management and test data analytics.

Using API functions you can view configured applications and test benches, run the configured test, and view test execution results.

Prerequisites for Clarius SDK installation

Install Clarius SDK in the oscilloscope or computer where the Clarius instrument service is installed. Install the following before you proceed with the Clarius SDK installation.

- Python 3.9 to 3.12. *Click here* to download.
- Install Clarius instrument service.
  - In the target system where the Clarius automation framework is installed, navigate to the installed path. The default path is `C:\Program Files\Tektronix\Clarius`.
  - Open Installers folder.
  - To install in an oscilloscope or computer, select and copy the Instrument folder.
  - Open Instrument folder, double-click **clarius-instrument-service-<<version>>.exe** and follow the steps to complete the installation.

The installation path for Clarius instrument service:

- If Clarius instrument service is installed in a computer or oscilloscope, then the installation path is `C:\Program Files\Tektronix\Clarius`.
- If Clarius instrument service is installed in the target system, then the installation path will be same as of Clarius automation framework.

Clarius SDK

Install Clarius SDK (Software Development Kit) in the target system (where Clarius automation framework is installed) or in an oscilloscope or computer where you need to run the automation script(s).

Clarius SDK can be installed in the following ways:

- Install Python in the global environment and then install Clarius SDK in that environment. If a supported Python version is detected, you can select to install the Clarius SDK in that environment.
- Install Python in an isolated Python environment[5] and install Clarius SDK in that environment.

Install Clarius SDK

If you have skipped Clarius SDK installation during the installation of Clarius automation framework, follow the steps to install.

1. In the target system, where the Clarius automation framework is installed, navigate to the installed path. The default path is `C:\Program Files\Tektronix\Clarius\installers\sdk`.

[5] An isolated Python environment will have its own independent set of Python packages installed in its site directories.

2. Select and copy the **sdk** folder and paste it to the oscilloscope or computer.
3. Open sdk folder, double-click **clarius-sdk-<<version>>.exe** and follow the steps to complete the installation.

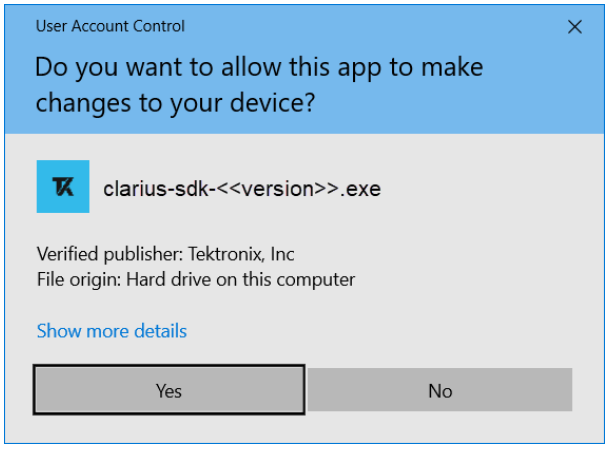


Figure 1: User account control dialog

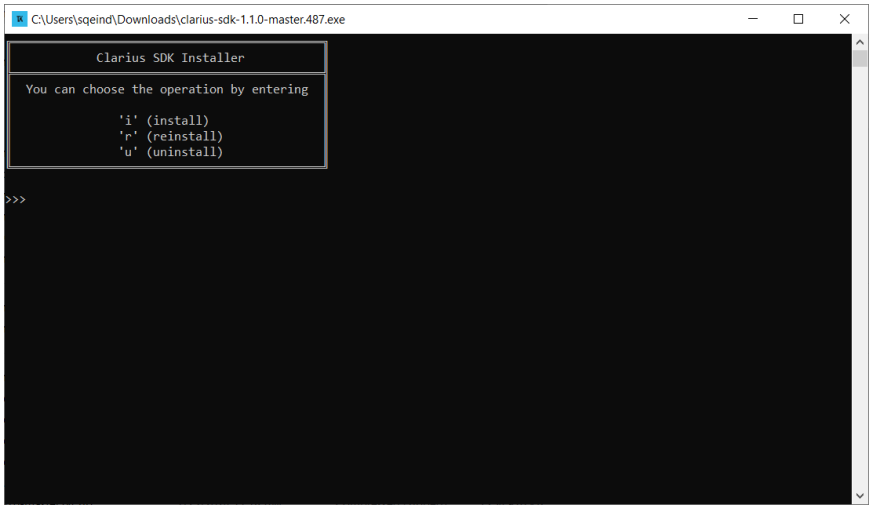


Figure 2: SDK installer setup

SDK function changelogs

This section lists the newly added SDK functions, modifications in the existing SDK functions and deprecated SDK functions from previous release.

Newly added functions

- *Delete multiple tests*
- *Download test waveforms larger than 1 GB*
- *Delete waveforms of test* on page 158

Changes in existing functions

Initialization	If the SSL certificate port and API port is changed during the installation of the Clarius automation framework, these ports must be configured during the initialization of Clarius SDK.
Create a new test	Acquisition mode and waveform path in “RECORDED” mode is moved from the application level to test level.
Set acquisition mode	
Create a new test bench	We can create new test bench based on acquisition mode and technologies/applications. **Note:** An internal Clarius test bench (Clarius_PC) will be created automatically on the computer/laptop where Clarius instrument service is installed.
Get interrupt notifications	• Interrupt actions data type is changed from string to enum. • You can handle test notification and system notification separately. Refer *Perform interrupt action* on page 172 for example scripts on handling test notifications and system notifications.
Perform interrupt action	

Deprecated functions

No functions are deprecated.

Importing and initializing Clarius SDK functions

The following example script imports and initializes the API, creates an Clarius api instance **api** and sets the response. You can use this instance to execute Clarius functions.

```
# Import Clarius SDK APIs
from clariussdk import clarius

# Initialization of API

ip = "<<Host IP>>"
client_id = "<<As Configured>>"
client_secret = "<<As Configured>>"

api_port = 8443
cert_port = 8080

# create an instance of Clarius API
api = clarius.Api(ip, client_id, client_secret, api_port, cert_port)
```

Table 63: Arguments

Argument Name	Argument Value
<ip>	ip address/host address of the Clarius VM (Virtual Machine).

Table continued…

Argument Name	Argument Value
<client_id>	id of the client. "<<As configured>>"
<client_secret>	Secret of the client. "<<As configured>>"
<api_port>	The default port is 8443. If the port was changed during the installation of Clarius automation framework, then please specify the configured port. The SSL Key port specified in portconfiguration.json is the configured port for api_port. The default file path is `C:\Program Files\Tektronix\Clarius\conf\`.
<cert_port>	The default port is 8080. If the port was changed during the installation of Clarius automation framework, then please specify the configured port. The apigateway port specified in portconfiguration.json is the configured port for cert_port.The default file path is `C:\Program Files\Tektronix\Clarius\conf\`

Get licensed applications

To get the licensed applications, use the following function. To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

```
apps = api.applications.get_list()
```

Returns

Returns licensed applications list.

```
[
    {
        "name":"",
        "description":"",
        "id":"",
        "category_type":"",
        "category_subtype":"",
        "execution_mode":""
    }
]
```

Test bench management

This section lists the functions that are related to Test bench management. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

Create a new test bench

To create a new test bench, use the following function.

```
api.test_benches.create_testbench(test_bench)
```

Arguments

Argument Name	Data Type	Argument Value
<test_bench>	custom JSON string	Test bench details in JSON single line.

Table 64: Test bench object for live acquisition

```
{
 "id": "string",
 "name": "string",
 "description": "string",
 "internal": bool,
 "validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA"
 "acquisitionMode": "LIVE",
 "technologies": [
  "string"
 ],
 "applications": [
  "string"
 ],
 "hubAddress": "http://<ip address>:18000",
 "instruments": [
  {
   "id": "string",
   "name": "string",
   "category": "string",
   "address": "string",
   "description": "string",
   "properties": {
    "additionalProp1": {},
    "additionalProp2": {},
    "additionalProp3": {}
   },
   "extensions": [
    {
     "id": "string",
     "name": "string",
```

```
            "category": "string",
            "address": "string",
            "description": "string",
            "properties": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            }
          }
        ]
      }
    ],
    "availability": "AVAILABLE"
  }
```

Note: The arguments in bold font are mandatory.

Table 65: Test bench object for recorded waveform

```
{
   "id": "string"
  "name": "string",
  "description": "string",
  "hubAddress": "http://<ip_address>:18000",
  "acquisitionMode": "RECORDED",
  "internal": false
}
```

Note:

- The arguments in bold font are mandatory.
-

Returns

200: Test bench updated successfully.

400: The validation of the request failed.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

404: The endpoint cannot be reached.

500: The system encountered an error and the user has to contact the administrator to resolve the problem.

Update the test bench of a test

To update the details of an existing test bench, use the following function.

```
api.test_benches.update_testbench(testbench_id,test_bench)
```

Arguments

Argument Name	Data Type	Argument Value
<testbench_id>	string	Name or id of the test bench for which you want to update the details in string format.
<test_bench>	Custom JSON string	Update/modify test bench details in the form of JSON in single line.

Table 66: Test bench object for live acquisition

```
{
 "id": "string",
 "name": "string",
 "description": "string",
 "internal": bool,
 "validationStatus": "NOT_VALIDATED| SUCCESS|FAILED|NA"
 "acquisitionMode": "LIVE",
 "technologies": [
   "string"
 ],
 "applications": [
   "string"
 ],
 "hubAddress": "http://<ip address>:18000",
 "instruments": [
   {
    "id": "string",
    "name": "string",
    "category": "string",
    "address": "string",
    "description": "string",
    "properties": {
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    },
    "extensions": [
      {
       "id": "string",
       "name": "string",
       "category": "string",
       "address": "string",
       "description": "string",
       "properties": {
         "additionalProp1": {},
         "additionalProp2": {},
         "additionalProp3": {}
       }
      }
    ]
   }
```

```
  ],
  "availability": "AVAILABLE"
}
```

Note: The arguments in bold font are mandatory.

Table 67: Test bench object for recorded waveform

```
{
   "id": "string"
   "name": "string",
   "description": "string",
   "hubAddress": "http://<ip_address>:18000",
   "acquisitionMode": "RECORDED",
   "internal": false
}
```

Note: The arguments in bold font are mandatory.

Returns

200: Test bench updated successfully.

400: The validation of the request failed.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

404: The endpoint cannot be reached.

500: The system encountered an error and the user has to contact the administrator to resolve the problem.

Get test bench details

To get specified test bench details, use the following function.

```
test_bench = api.test_benches.get_testbench(testbench_id)
```

Arguments

Argument Name	Data Type	Argument Value
<testbench_id>	string	Name or id of the test bench for which you want to get the details in string format.

Returns

Returns details of the specified test bench id. If the specified test bench id is not present, then returns a 404 error.

```
{
  "id": "",
  "name": "",
  "description": "",
  "technologies": [""],
  "hubAddress": "",
  "instruments": [
    {
      "id": "",
      "name": "",
      "type": "",
      "category": "",
      "address": "",
      "description": "",
      "properties": {
        "additionalProp1": {},
        "additionalProp2": {},
        "additionalProp3": {}
      },
      "extensions": [
        {
          "id": "",
          "name": "",
          "type": "",
          "category": "",
          "address": "",
          "description": "",
          "properties": {
            "additionalProp1": {},
            "additionalProp2": {},
            "additionalProp3": {}
          }
        }
      ]
    }
  ],
  "availability": "AVAILABLE",
  "lastValidatedDateTime": "DD/MM/YYYY-HH:MM:SS",
  "validationStatus": ""
}
```

Get all configured test benches

To get all test benches configured in the Clarius application, use the following function.

```
test_benches = api.test_benches.get_list()
```

Returns

Returns list of all configured test benches.

Table 68: Response

```
[
  {
    "id": "",
    "name": "",
    "description": "",
    "technologies": [""],
    "hubAddress": "",
    "instruments": [
      {
        "id": "",
        "name": "",
        "type": "",
        "category": "",
        "address": "",
        "description": "",
        "properties": {
          "additionalProp1": {},
          "additionalProp2": {},
          "additionalProp3": {}
        },
        "extensions": [
          {
            "id": "",
            "name": "",
            "type": "",
            "category": "",
            "address": "",
            "description": "",
            "properties": {
              "additionalProp1": {},
              "additionalProp2": {},
              "additionalProp3": {}
            }
          }
        ]
      }
    ],
    "availability": "AVAILABLE",
    "lastValidatedDateTime": "DD/MM/YYYY-HH:MM:SS",
    "validationStatus": ""
  }
]
```

Delete test bench

To delete an existing test bench, use the following function.

```
api.test_benches.delete_testbench(testbench_id)
```

Arguments

Argument Name	Data Type	Argument Value
<testbench_id>	string	Name or id of the test bench for which you want to get the details in string format.

Returns

204: Test bench deleted successfully.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

404: The endpoint cannot be reached.

500: The system encountered an error and the user has to contact the administrator to resolve the problem.

Create a new test

To create a new test use the following function. This function sets the response to the Clarius api instance **new_test**. You can use this instance to execute all test related functions.

To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

```
new_test = api.tests.new_test(test_name, test_bench_id, description)
```

Arguments

Argument Name	Data type	Argument Value
<test_name>	string	User-defined name for test.
<test_bench_id>	string	Name or id of the test bench to run a test. Use test bench name as *Clarius_PC*[6] to run the tests on the waveforms located in the target system where Clarius automation framework is installed.
<description>	string	User-defined description for a test.

Returns

Returns the new test object.

[6] It is an internal test bench created to run the measurements on the recorded waveforms in the target system where Clarius automation framework is installed.

```
{
  "test_name":"",
  "testbench_id":"",
  "description":"",
  "selected_apps":[]
}
```

Set acquisition mode

To set the acquisition mode, use the following function.

- Use this script to set acquisition mode as live

```
from clariussdk.tests import AcquisitionMode
new_test.acquisition_mode = AcquisitionMode.LIVE
```

- Use this script to set acquisition mode as pre-recorded waveform.

```
from clariussdk.tests import AcquisitionMode
new_test.acquisition_mode = AcquisitionMode.RECORDED
new_test.waveform_folder_path = "<base_waveform_path>"
```

Arguments

Argument Name	Data Type	Argument Value
<base_waveform_path>	string	Waveform path

Get supported technologies for a test

To get the supported technologies for a test, use the following function.

```
new_test.get_available_technologies()
```

Returns

Returns the list of supported technologies.

```
["Tx Tech1", "Tx Tech2"]
```

Get application(s) of a specific technology

To get the application(s) of a specific technology, use the following.

```
new_test.get_available_applications(technology)
```

Arguments

Argument Name	Data Type	Argument Value
<technology>	string	Technology name

Returns

Returns a list of applications of a specific technology.

```
["Application1", "Application2", "Application3"]
```

Get application information of a test

To get the application list use the following function.

```
new_test.get_applications_info()
```

Returns

Returns list of sequence information of a test.

```
[
  {
    "testCategory":"",
    "technology":"",
    "index":"",
    "name":"",
    "id":"",
    "selected":"true | false"
  }
]
```

Get technology information

Query the technology information of a test or sequence, using the following function. This function sets the response to the Clarius api instance **technology**. You can use this instance to execute all technology related functions. To execute this function, use the Clarius api instance **new_test** from *Create a new test* function.

```
technology = new_test.get_technology(testCategory, technology)
```

```
technology = sequence.get_technology(testCategory, technology)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category of a technology
<technology>	string	Technology name

Returns

Returns a list of technology information using applications that are configured for a specific test or sequence.

Set the selection status of an application in a test

To set the selection status of an application in a test, use the following function.

```
technology.applications[index].set_selection(selection_status)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<selection_status>	boolean	Boolean value can be True or False.

Get the selection status of an application in a test

To get the selection status of an application in a test, use the following function.

```
technology.applications[index].get_selection()
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….

Returns

Returns status of selection, either True or False.

Get global settings configured for an application in a test

To get the global settings configured for an application in a test, use the following function.

```
app_settings = technology.applications[index].global_settings
```

Arguments

Argument Name	Argument Value
<index>	Index of the application which starts with 0,1,2,3….

Returns

Returns application-specific global settings.

```
[
  {
    "referenceGroupBy":"",
    "name":"",
    "type":"STRING",
    "displayName":"",
    "group":"DUT",
    "reference":"",
    "referenceType":"",
    "value":"",
    "constraints":"value IN ("")",
    "internal":true | false,
    "unit":"MT/s",
    "description":"",
    "additionalProperties":"",
    "category":"",
    "tag":"",
    "global":true | false,
    "mandatory":true | false,
    "editable":true | false,
    "deprecated":true | false
  }
]
```

Get configured settings for a specified setting name of an application in a test

To get the configured settings for a specified setting name of an application in a test, use the following function.

```
app_setting = technology.applications[index].get_setting(setting_name)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<setting_name>	string	Name of the setting for which you want to get the details.

Returns

Setting name, value, constraints of the setting, and other details of the setting.

```
{
  "referenceGroupBy":"",
  "name":"",
  "type":"STRING",
  "displayName":"",
  "group":"Acquisition",
  "reference":"",
  "referenceType":"",
  "value":"",
  "constraints":"value IN ("")",
  "internal":true | false,
  "unit":"GHz",
  "description":"",
  "additionalProperties":"",
  "category":"",
  "tag":"",
  "global":true | false,
  "mandatory":true | false,
  "editable":true | false,
  "deprecated":true | false
}
```

Set application global settings

To set application global settings, use the following function.

```
technology.applications[index].set_setting(setting_name, setting_value)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<setting_name>	string	Name of the global settings to which you want to set the value.

Table continued…

Argument Name	Data Type	Argument Value
<setting_value>	Standard or custom data type depending on the value of the setting.	Value to set.

Get specified application scenarios of a test

To get the specified application scenarios of a test, use the following function.

```
list_scenario = technology.applications[index].list_scenarios()
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….

Returns

Returns the list of the scenarios.

```
[
   {
      "scenarioName":"",
      "description":""
   }
]
```

Select a specified application scenario in a test

To select a specified application scenario in a test, use the following function.

```
technology.applications[index].select_scenario(scenario_name)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario that you want to select.

Unselect a specified application scenario in a test

To unselect a specified application scenario in a test, use the following function.

```
technology.applications[index].unselect_scenario(scenario_name)
```

Arguments

Argument Name	Data Type	Argument Value
<scenario_name>	string	Name of the application scenario that you want to unselect.

Get list of settings for a specific scenario

To get the list of settings for a specific scenario, use the following function.

```
list_scenario_settings =
technology.applications[index].get_scenario_setting(scenario_name)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario that you want to get settings.

Returns

Returns scenario settings.

```
[
  {
    "referenceGroupBy":"",
    "name":"",
    "type":"",
    "displayName":"",
    "group":"",
    "reference":"",
    "referenceType":"",
    "value":{},
    "constraints":"value IN ("" AND value != "" AND value.Count > ",
    "internal":true | false,
    "unit":"",
    "description":"",
    "additionalProperties":"",
    "category":"",
    "tag":"",
    "global":true | false,
    "mandatory":true | false,
    "editable":true | false,
    "deprecated":true | false
```

```
    }
]
```

Set value for a specific settings of an scenario

To set the value for a specific settings of an scenario, use the following function.

```
technology.applications[index].set_scenario_setting(scenario_name,
setting_name,setting_value)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario for which you want to get settings.
<setting_name>	string	Name of the scenario setting.
<setting_value>	Standard or custom data type depending on the value of the setting.	Value which you want to set.

Get list of steps available in a scenario

To get the list of steps available in a scenario, use the following function.

```
list_step = technology.applications[index].list_steps(scenario_name)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario for which you want to get steps.

Returns

Returns the list of steps available in scenario settings.

```
[
    {
        "stepName":"",
        "description":""
```

```
    }
]
```

Get list of settings available for a step

To get the list of settings available for a step in a scenario, use the following function.

```
step_settings =
technology.applications[index].get_step_setting(scenario_name,step_name)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario for which you want to get settings.
<step_name>	string	Name of the application step for which you want to get settings.

Returns

Returns list of specified step settings

```
[
  {
    "referenceGroupBy":"",
    "name":"",
    "type":"",
    "displayName":"",
    "group":"",
    "reference":"",
    "referenceType":"",
    "value":[
      {
        "name":"",
        "min":"",
        "max":""
      }
     ],
    "constraints":"",
    "internal":true | false,
    "unit":"",
    "description":"",
    "additionalProperties":"",
    "category":"",
    "tag":"",
    "global":true | false,
    "mandatory":true | false,
```

```
        "editable":true | false,
        "deprecated":true | false
      }
    ]
```

Set value for a specific setting in a step

To set value for a specific setting in a step, use the following function.

```
technology.applications[index].set_step_setting(scenario_name,
step_name,setting_name,setting_value)
```

Arguments

Argument Name	Data Type	Argument Value
<index>	int	Index of the application which starts with 0,1,2,3….
<scenario_name>	string	Name of the application scenario for which you want to get settings.
<step_name>	string	Name of the application step for which you want to get settings.
<setting_name>	string	Name of the step setting to which you want to set value.
<setting_value>	Standard or custom data type depending on the value of the setting.	Value which you want to set.

Signal sources

This section describes the functions to configure signal sources before starting a test. To execute this function, use the Clarius api instance **new_test** from *Create a new test* function.

View default signal sources of a technology

To view the default signal sources of a technology, use the following function.

```
new_test.get_technology(testCategory,
technology).view_default_signal_sources()
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to view the signal source.
<technology>	string	Technology name.

Returns

Returns the list of default sources and signals of a technology.

Update the existing signal source of a technology

To update the existing or new signal source of a technology, use the following function.

- Use this script to add a new source.

```
new_test.get_technology(testCategory,technology).add_source(source_name)
```

- Use this script to view all signals defined under a source.

```
new_test.get_technology(testCategory,
technology).view_all_signals(source_name)
```

- Use this script to add a signal to a source.

```
new_test.get_technology(testCategory, technology).add_signal(source_name,
signal_name, probe_method)
```

- Use this script to set the signal probe method.

```
new_test.get_technology(testCategory,
technology).set_signal_probe_method(source_name, signal_name, probe_method)
```

- Use this script to get the signal of a source.

```
signal = new_test.get_technology(testCategory,
technology).get_signal(source_name, signal_name)
```

- Use this script to update the signal of a source.

```
signal = new_test.get_technology(testCategory,
technology).get_signal(source_name, signal_name)
signal[index].update_signal(label, channel, instrument)
```

Note:

The value set for "channel" and "instrument" is considered only for live acquisition and not for recorded acquisition mode.

- Use this script to remove the signal of a source.

```
new_test.get_technology(testCategory,
technology).remove_signal(source_name, signal_name)
```

- Use this script to remove a source.

```
new_test.get_technology(testCategory,
technology).remove_source(source_name)
```

- Use this script to view all configured sources.

```
new_test.get_technology(testCategory, technology).view_configured_sources()
```

Arguments

Argument Name		Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to update the signal source.
<technology>	string	Technology name.
<source_name>	string	Name of the source such as Lane0, Clk.
<signal_name>	string	Name of the signal in a source such as DATA.
<probe_method>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Probing methods such as SINGLE ENDED/DIFFERENTIAL.
<index>	int	Index of the application which starts with 0,1,2,3….
<label>[7]	string	User-defined signal label.
<channel>	string	Name of the channel to which the signal is connected such as CH1, CH3.
<instrument>	string	Name of the instrument to which the signal is connected.

Returns

Returns the updated, existing, or newly added signal sources of a technology.

View the supported sources of a technology

To view the supported sources of a technology, use the following function.

```
new_test.get_technology(testCategory, technology).view_supported_sources()
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to view the signal source.
<technology>	string	Technology name.

[7] This argument is optional.

Returns

Returns the list of supported sources of a technology.

Remove the existing source of a technology

To remove the existing source of a technology, use the following function.

```
new_test.get_technology(testCategory, technology).remove_source(source_name)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to remove the signal source.
<technology>	string	Technology name.
<source_name>	string	Name of the source such as Lane0, Clk.

Get all selected lanes

To get all selected lanes, use the following function.

```
new_test.get_technology(testCategory, technology).get_selected_lanes()
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to get all lanes.
<technology>	string	Technology name.

Returns

Returns the list of all selected lanes.

```
[
    "Lane0",
    "Lane1"
]
```

Get all selected lane groups

To get all selected lane groups, use the following function.

```
new_test.get_technology(testCategory, technology).get_lane_groups()
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to get all lanes.
<technology>	string	Technology name.

Returns

Returns the list of all selected lane groups.

```
[
  {
    "Group":"",
    "Lanes":""
  }
]
```

Add a lane group

To add a lane group, use the following function.

```
new_test.get_technology(testCategory, technology).add_lane_group(lanes)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to add a lane group.
<technology>	string	Technology name.
<lanes>	string	Select the lanes such as Lane0, Lane1.

Update a lane group

To update a lane group, use the following function.

```
new_test.get_technology(testCategory,
technology).update_lane_group(group_name, lanes)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to update the lane group.
<technology>	string	Technology name.
<group_name>	string	Name of the lane group.
<lanes>	string	Select the lanes such as Lane0, Lane1.

Returns

Returns the updated lane group.

Remove a lane group

To remove a lane group from the technology, use the following function.

```
new_test.get_technology(testCategory,
technology).remove_lane_group(group_name)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to remove a lane group.
<technology>	string	Technology name.
<group_name>	string	Name of the lane group.

Returns

Removes the lane group from the technology.

Measurement limits

This section lists the functions to configure measurement limits of an application and edit the limits before starting a test. To execute this function, use the Clarius api instance **new_test** from *Create a new test* function.

Get all measurement limits of an application

To get all measurement limits of an application, use the following function.

```
limits = new_test.get_technology(testCategory,
technology).applications[index].get_limits()
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category of a technology.
<technology>	string	Technology name.
<index>	int	Index of the application which starts with 0,1,2,3….

Returns

Returns the list of limits in the application.

```
[
  {
    MeasurementLimit(name="",
    displayName="",
    group="",
    idealValue="",
    lowLimit=Limit(value="", comparator=">="),
    highLimit=Limit(value="", comparator="<="),
    unit="V",
    additionalInfo={"DataRate": "<= 1"}),
  }
]
```

Get specific measurement limit of an application

To get a specific measurement limit of an application, use the following function.

```
limit = new_test.get_technology(testCategory,
technology).applications[index].get_limit(limit_name, group_name,
additional_info)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category of a technology.
<technology>	string	Technology name.
<index>	int	Index of the application which starts with 0,1,2,3….
<limit_name>	string	Name of the measurement.
<group_name>[7]	string	Measurement group.
<additional_info>[7]	dict	Additional metadata.

Returns

Returns the specific measurement limit of an application.

```
{
    name=""
    displayName=""
    group="" idealValue=""
    lowLimit=Limit(value="", comparator=">=")
    highLimit=Limit(value="", comparator="<=")
    unit="V"
    additionalInfo={"DataRate": "<= 1"}
  }
```

Update the measurement limit of an application

To update the measurement limit of an application, use the following function.

- Use this script to update the limit's ideal value.

```
limit.update_ideal_value(259.24)
```

- Use this script to update lower-limit information.

```
limit.update_low_limit(205.32, ">")
```

- Use this script to update higher-limit information.

```
limit.update_high_limit(309.08, "<")
```

Arguments

Argument Name	Data Type	Argument Value
update_ideal_value	float	Expected ideal value of the measurement result.
update_low_limit (value, comparator)	value: float, comparator: string	Defines low limit for the measurement result. Comparator values: = =, ! =, > =, >
update_high_limit (value, comparator)	value: float, comparator: string	Defines high limit for the measurement result. Comparator values:< =, <

Start a new test

Prerequisites

- *Initialize the SDK*
- *Create test bench and set the test bench parameters*
- *Create test by specifying the name and technology; Add application to the test and configure the technology settings*

To start a new test, use the following function. To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

```
execution_id = api.tests.start_test(new_test)
```

Arguments

Argument Name	Data Type	Argument Value
<new_test>	custom	Test which you want to start.

Returns

Returns execution_id.

Get test status from the application

To get the status of a test, use the following function.

- Use this script to get the test execution status summary.

```
test = api.tests.get_status(execution_id)
```

- Use this script to get the test execution status in detail.

```
test = api.tests.get_status(execution_id, raw=True)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id.
<raw>	boolean	It is a boolean value to get the details of the test status. Either "raw=True" or "raw=False".

Returns

Returns the status of the test execution.

For raw=False:

```
{
  "id":"",
  "test_name":"",
  "start_time":"YYYY-MM-DDTHH:MM:SS",
  "end_time":"YYYY-MM-DDTHH:MM:SS",
  "duration":"HH:MM:SS",
  "executed_scenarios":"",
  "status":"PASSED/FAILED"
}
```

For raw=True:

```
{
  "executionId":"",
  "executionName":"",
  "linkId":"",
  "tags":[
    "default"
  ],
  "description":"",
  "executionMode":"COMPLIANCE | USER | CHARACTERIZATION",
  "testBenchId":"",
  "totalScenarios":"",
  "status":"PASSED/FAILED",
  "executedScenarios":"",
  "startTime":"YYYY-MM-DDTHH:MM:SS",
  "endTime":"YYYY-MM-DDTHH:MM:SS",
  "testStatuses":[
    {
      "tag":"",
      "applicationStatuses":[
        {
          "selectionId":"",
          "applicationId":"",
          "applicationName":"",
          "applicationDescription":"",
          "scenarioStatuses":[
            {
              "name":"",
              "id":"",
              "status":"PASSED/FAILED",
              "startTime":"YYYY-MM-DDTHH:MM:SS",
              "endTime":"YYYY-MM-DDTHH:MM:SS",
              "stepStatuses":[
                {
                  "additionalProperties":{},
                  "name":"",
                  "id":"",
                  "status":"PASSED/FAILED",
                  "startTime":"YYYY-MM-DDTHH:MM:SS",
```

```
                        "endTime":"YYYY-MM-DDTHH:MM:SS"
                      }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
```

Get all test executions

To get a list of executed tests, use the following function.

```
tests = api.tests.get_list(raw,page,page_size)
```

Arguments

All arguments are optional. If the values are not passed for the arguments, then the function returns latest 30 test lists (saved and executed).

Argument Name	Data Type	Argument Value
<raw>	boolean	Set True to get comprehensive details.
<page>	int	Select the page number from where to get the executed tests.
<page_size>	int	Specify the total tests to add in a page. Range: 1 to 30 (Default: 30)

Returns

This function returns the list of all executed tests.

For raw=False:

```
[
  {
    "id":"",
    "test_name":"",
    "start_time":"YYYY-MM-DDTHH:MM:SS",
    "end_time":"YYYY-MM-DDTHH:MM:SS",
    "duration":"HH:MM:SS",
    "executed_scenarios":"",
    "status":"PASSED/FAILED"
  }
]
```

For raw=True:


```
[
{
  "executionId": "",
  "executionName": "",
  "linkId": "",
  "tags": ["DEFAULT"],
  "applicationRequests": [
    {
      "technology": "",
      "testCategory": {
        "type": "TX",
        "subType": ""
      },
      "loop": {
        "sources": [
          [
            {
              "name": "",
              "type": "",
              "signals": [
                {
                  "name": "",
                  "category": "",
                  "probeMethod": "",
                  "singleEnded": ,
                  "differential": [
                    {
                      "name": "",
                      "label": "",
                      "type": "",
                      "polarity": ,
                      "channel": "",
                      "instrument": ""
                    }
                  ]
                }
              ]
            },
            {
              "name": "",
              "type": "",
              "signals": [
                {
                  "name": "",
                  "category": "",
                  "probeMethod": "",
                  "singleEnded": ,
                  "differential": [
                    {
                      "name": "",
```

```
                    "label": "",
                    "type": "",
                    "polarity": ,
                    "channel": "",
                    "instrument": ""
                  }
                ]
              }
            ]
          }
        ]
      ]
    },
    "applicationId": ""
  },
  "tag": "",
  "settings": [],
  "applicationSelections": [
    {
      "settings": [],
      "selectionId": "",
      "applicationId": "",
      "selected": true,
      "scenarioNames": [""],
      "scenarioSelectionInfo": [
        {
          "scenarioName": "",
          "selected":
        }
      ]
    }
  ]
],
"description": "",
"executionMode": "",
"acquisitionMode": "",
"testBenchId": "",
"status": "",
"totalScenarios": ,
"startTime": ""YYYY-MM-DDTHH:MM:SS"",
"endTime": ""YYYY-MM-DDTHH:MM:SS"",
"executedScenarios": ,
"testbenchDetails": {
  "name": "",
  "instrumentInfo": [
    {
      "_id": "",
      "address": "",
      "category": "",
      "categoryDisplayName": "",
```

```
            "description": "",
            "name": "",
            "properties": {
              "manufacturer": "",
              "model": "",
              "serialNo": "",
              "firmwareVersion": "",
              "calibration": [
                {
                  "spcStatus": "",
                  "spcPerformedDate": ""
                }
              ],
              "probes": [
                {
                  "channel": "",
                  "serialNo": "",
                  "probeType": "",
                  "tipType": "",
                  "calibrationStatus": ""
                },
                {
                  "channel": "",
                  "serialNo": "",
                  "probeType": "",
                  "tipType": "",
                  "calibrationStatus": ""
                },
                {
                  "channel": "",
                  "serialNo": "",
                  "probeType": "",
                  "tipType": "",
                  "calibrationStatus": ""
                },
                {
                  "channel": "",
                  "serialNo": "",
                  "probeType": "",
                  "tipType": "",
                  "calibrationStatus": ""
                }
              ],
              "deskew": [
                {
                  "channel": "",
                  "deskew": ""
                },
                {
                  "channel": "",
```

```
                "deskew": ""
              },
              {
                "channel": "",
                "deskew": ""
              },
              {
                "channel": "",
                "deskew": ""
              }
            ]
          }
        }
      ],
      "id": ""
    },
    "softwareInfo": {
      "MultiLaneApp_4lane": "",
      "clariusVersion": ""
    }
  }
  ]
```


Get filtered test execution list

To get a list of executed tests, by applying filters use the following function.

```
tests = api.tests.get_filtered_list(test_name, application_names, page,
page_size, status, from_date, to_date)
```

Arguments

All arguments are optional. If the values are not passed for the arguments, then the function returns latest 30 test lists (saved and executed).

Arguments name	Data Type	Description
test_name	str	Name of the test to filter
application_names	list[str]	Names of the application to filter
page	int	Specify the page number to get the tests list of that page. Default: 1
page_size	int	Specify the total test list to display in single page. Default: 30

Table continued…

Arguments name	Data Type	Description
status	list[str]	Test status to filter. The values are PASSED, FAILED, DRAFT, RUNNING, ABORTED.
from_date	str	Select the from date (YYYY:MM:DD) to filter the tests.
to_date	str	Select the to date (YYYY:MM:DD) to filter the tests.

Returns

This function returns the list of tests that matches the filter condition.

```
[
  {
    "id":"",
    "test_name":"",
    "start_time":"YYYY-MM-DDTHH:MM:SS",
    "end_time":"YYYY-MM-DDTHH:MM:SS",
    "duration":"HH:MM:SS",
    "executed_scenarios":"",
    "status":"PASSED/FAILED"
  }
]
```

Test results management

This section lists the functions that are related to Test results management. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

Get results of an executed test

To get the test results of an executed test, use the following function.

```
results = api.tests.get_results(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id

Returns

Test execution results.

```
{
  "executionId":"",
  "executionName":"",
```

```
"linkId":"",
"tags":[
  "default"
],
"description":"",
"testResults":[
  {
    "tag":"DEFAULT",
    "applicationResults":[
      {
        "selectionId":"",
        "applicationId":"",
        "settings":{},
        "scenarioResults":[
          {
            "name":"",
            "id":"",
            "waveforms":[
              {
                "waveformURI":"/",
                "waveformMetadata":{
                  "id":"",
                  "name":"",
                  "iterationNumber":"",
                  "source":{
                    "channel":"",
                    "type":"SINGLE/DIFFERENTIAL",
                    "signal":"",
                    "polarity":"",
                    "mathDefinition":""
                  },
                  "setup":{
                    "name":""
                  },
                  "additionalProperties":{
                    "prefix":""
                  }
                }
              }
            ],
            "status":"",
            "analysisResults":[
              {
                "errors":"",
                "additionalProperties":{
                },
                "settings":[
                  {
                    "referenceGroupBy":"",
                    "name":"",
                    "type":"",
```

```
                        "displayName":"",
                        "group":"",
                        "reference":"",
                        "referenceType":"",
                        "value":[
                          {
                            "name":"",
                            "instrument":"Scope",
                            "channel":"",
                            "type":"SINGLE_ENDED/DIFFERENTIAL",
                            "signal":""
                          }
                                ],
                        "constraints":"",
                        "internal":true | false,
                        "unit":"",
                        "description":"",
                        "additionalProperties":"",
                        "category":"",
                        "tag":"",
                        "global":true | false,
                        "mandatory":true | false,
                        "editable":true | false,
                        "deprecated":true | false
                      },
                    "attachments":[
                      {
                        "name":"",
                        "description":"",
                        "contentType":"application/json",
                        "uri":"",
                        "additionalProperties":{
                          "content-format":"json",
                          "filename":""
                        }
                      }
                    ],
                    "status":"",
                    "statusDescription":""
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
    "status":"",
    "startTime":"YYYY-MM-DDTHH:MM:SS",
    "endTime":"YYYY-MM-DDTHH:MM:SS",
    "executionMode":"",
```

```
  "duration":"HH:MM:SS"
}
```

Get results of a specific step

To get the results of a specific step under a scenario, use the following function.

```
results =
api.tests.get_step_results(execution_id,app_id,scenario_name,step_name)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id
<app_id>	string	Application id
<scenario_name>	string	Name of the scenario
<step_name>	string	Name of the step

Returns

Returns result of a specified step.

```
[
  {
    "iterationNumber":"",
    "results":[
      [
        {
          "name":"",
          "type":"",
          "value":"",
          "constraints":"",
          "internal":true | false,
          "unit":"ps",
          "additionalProperties":{
            "marginValue":"",
            "resultStatus":"Informative | Pass"
        }
        }
      ]
    ]
  }
]
```

Get analysis results of an executed test

To get the analysis results of an executed test, use the following function.

```
result = api.tests.get_analysis_results(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id

Returns

Returns the analysis result of a test execution in the form of two-dimensional data table.

Columns of the result can be: application, scenario, step, iteration, acquisition, device_state, measurement, units, group, value, mean, min, max, std_dev, peak_peak, count, status, limits, lower_margin, higher_margin, remarks, info.

Get cumulative analysis results of an executed test

To get the cumulative analysis results of an executed test, use the following.

```
result = api.tests.get_analysis_results(execution_id,cumulative=True)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id.
<cumulative>	boolean	It is a boolean value that indicates if the user wants cumulative analysis result. Either "cumulative=True" or "cumulative=False".

Returns

Returns the cumulative analysis result of a test execution in the form of two-dimensional data table.

Columns of the result can be: application, scenario, step, device_state, measurement, units, group, value, mean, min, max, std_dev, peak_peak, count, status, limits, lower_margin, higher_margin, remarks, info.

Download test waveform

To download waveforms of a test, application, scenario or step, use the following function. To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

- Use this script to download waveforms of size less than 1 GB

```
api.results.download_waveforms(execution_id, application_id,
scenario_name, step_name, wfm_path)
```

- Use this script to download waveforms of size greater than 1 GB or multiple waveforms

```
api.waveforms.download(execution_id, application_id, scenario_name,
step_name, wfm_path)
```

Arguments

Argument Name	Data Type	Argument Value
<execution id>	string	Test execution id
<application id>[8]	string	Application id
<scenario name>[8]	string	Scenario name
<step name>[8]	string	Step name
<wfm_path>[9]	string	Waveform download path

Returns

It downloads the waveform of a test, application, scenario or step.

Abort a test

To abort a test, use the following.

```
api.tests.abort_test(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id

Wait for test completion

To wait until the test execution is complete, use the following.

Note: This is a blocking call. SDK execution will wait till test execution is complete for timeout seconds.

[8] This argument is optional. If the child argument (For example: scenario name) is given then the parent argument (For example: application id) is mandatory.
[9] This argument is optional. If the path is not provided then the waveforms will be downloaded in the current working directory.

```
api.tests.wait_for_completion(execution_id,timeout=3000,delay=1)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id
<timeout>	int	Wait time in seconds
<delay>	int	Delay for wait time in seconds

Delete a test

To delete a test, use the following function.

```
api.tests.delete_test(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id

Delete multiple tests

To delete multiple tests, use the following function.

```
execution_id_list = ["testId1", "testId2", "testId3"]
api.tests.delete_tests(execution_id_list)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id_list>	string	test id list

Delete waveforms of test

To delete the waveforms of test, use the following function.

```
execution_id_list = ["testId1", "testId2", "testId3"]
api.results.delete_waveforms(execution_id_list)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id_list>	string	Execution id list

Test events

This section lists the functions related to test events. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

Get test events

To get the test events of an executed test, use the following function.

```
result = api.logs.get_test_events(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id.

Returns

Returns the test events of the specified execution id.

```
{
    "id": "",
    "level": "EVENT",
    "logTime": "YYYY-MM-DDTHH:MM:SS",
    "host": "",
    "service": "",
    "component": "",
    "transactionId": "",
    "transactionType": "",
    "additionalTags": {},
    "source": {
        "processId": "",
        "threadId": "",
        "name": "",
        "location": ""
```

```
        },
        "message": ""
    }
```

Get test events based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, page, pageSize, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
result = api.logs.get_test_events_params(query_param_dict)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	Example: query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
<from>	string	Specify the date and time to fetch the logs based on from or to values in the format YYYY-MM-DD HH:MM.
<to>	string	
<page>	int	Specify the page number to fetch the log of that particular page.
<pageSize>	int	Specify the page size to fetch the logs based on it. It is like the number of pages that you want to retrieve.
<origin>	string	Specifies the origin of the transaction request.
<host>	string	Specifies the host on which the service is running.
<service>	string	Specifies the service that is logging the message.
<component>	string	Specifies the component of the service which is logging the message.
<transactionId>	string	Specifies the unique id of transaction during which the message was logged.
<transactionType>	Custom strings. Can take only 2 values (TEST, RESOURCE)	Specifies the type of transaction during which message was logged.

Returns

Returns events of the test based on specified query parameter.

```
    {
        "id": "",
        "level": "EVENT",
        "logTime": "YYYY-MM-DDTHH:MM:SS",
        "host": "",
```

```
        "service": "",
        "component": "",
        "transactionId": "",
        "transactionType": "",
        "additionalTags": {},
        "source": {
            "processId": "",
            "threadId": "",
            "name": "",
            "location": ""
        },
        "message": ""
    }
```

Download test events based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, page, pageSize, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
api.logs.download_test_events(query_param_dict)
api.logs.download_test_events(query_param_dict, download_folder_path)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	A variable that is defined in the *function*.
<download_folder_path>	string	This method is used to download the logs of the test based on the specified above query parameters in dictionary format under the specified folder or else by default under the current working directory with a folder name called test_logs in the form of JSON.

Returns

Test event JSON file.

Delete test events based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
api.logs.delete_test_events(query_param_dict)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	Delete the events of the test based on above mentioned query parameter in the dictionary format.

Returns

204: Test events deleted successfully.

400: The validation of query parameters failed.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

500: System errors encountered. The user here has to contact the administrator to rectify the problem.

Test logs

This section lists the functions related to test logs. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

Get test logs

To get the test logs of an executed test, use the following function.

```
result = api.logs.get_test_logs(execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<execution_id>	string	Test execution id. **Example:** "a08a7d91-9730-43f8-9deb-adf2a6d0b619"

Returns

Returns the logs of the specified execution id.

```
{
  "id":"",
  "level":"INFO/ERROR",
  "logTime":"YYYY-MM-DDTHH:MM:SS",
  "host":"<<ComputerDevicename>>",
  "service":"Instrument_Service/Analysis_Service",
  "additionalTags":{

  },
  "source":{
```

```
        "processId":"",
        "threadId":"",
        "name":"",
        "location":""
      },
      "message":""
    }
```

Get test logs based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, page, pageSize, level, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
result = api.logs.get_test_logs_params(query_param_dict)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	Example: query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
<from>	string	Specify the date and time to fetch the logs based on from or to values in the format YYYY-MM-DD HH:MM.
<to>	string	
<page>	int	Specify the page number to fetch the log of that particular page.
<pageSize>	int	Specify the page size to fetch the logs based on it. It is like the number of pages that you want to retrieve.
<level>	Custom string. Can only take the specified value.	This specifies the message, such as: • ERROR • INFO • WARNING
<origin>	string	Specifies the origin of the transaction request.
<host>	string	Specifies the host on which the service is running.
<service>	string	Specifies the service that is logging the message.
<component>	string	Specifies the component of the service which is logging the message.
<transactionId>	string	Specifies the unique id of the transaction during which the message was logged.
<transactionType>	Custom string. Can take only 2 values (TEST, RESOURCE).	Specifies the type of transaction during which the message was logged.

Returns

Returns logs of the test for the specified query parameter.

```
[
  {
    "id":"",
    "level":"INFO/ERROR",
    "logTime":"YYYY-MM-DDTHH:MM:SS",
    "host":"<<ComputerDevicename>>",
    "service":"Instrument_Service/Analysis_Service",
    "component":"",
    "additionalTags":{

    },
    "source":{
      "processId":"",
      "threadId":"",
      "name":"",
      "location":""
    },
    "message":""
    "trace":""
  }
]
```

Download test logs based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, page, pageSize, level, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
api.logs.download_test_logs(query_param_dict)
api.logs.download_test_logs(query_param_dict, download_folder_path)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	A variable that is defined in the *function*.
<download_folder_path>	string	This method is used to download the logs of the test based on the specified above query parameters in dictionary format under the specified folder or else by default under the current working directory with a folder name called test_logs in the form of JSON.

Returns

Test log JSON file.

Delete test logs based on query parameters

Query parameters are passed as a dictionary. The list of supported query parameters are: **from, to, level, origin, host, service, component, transactionId, and transactionType**.

```
query_param_dict = {"from":"YYYY-MM-DD HH:MM", "service":"Instrument_Service"}
api.logs.delete_test_logs(query_param_dict)
```

Arguments

Argument Name	Data Type	Argument Value
<query_param_dict>	dict	Delete the logs of the test based on above mentioned query parameter in dictionary format.

Returns

204: Test logs deleted successfully.

400: The validation of query parameters failed.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

500: System errors encountered. The user here has to contact the administrator to rectify the problem.

Reports

This section lists the functions related to generating report of a test. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

Get list of report templates

To get the list of report templates, use the following function.

```
templates = api.templates.get_report_templates()
```

Returns

Returns the list of available report templates.

Customize and generate report of a test

To generate the report of a test at a specified path, use the following function. To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

```
#(Optional)Set the report settings; this will configure the data to include
in the report
api.reports.customize_report_generation(include_plots, include_waveforms,
include_testbench,include_test_configuration, user_comment, logo_path)

#Generate report
api.reports.generate_report(test_id,template_id,report_name,report_path)
```

Arguments

Argument Name	Data Type	Argument Value
include_plots	Boolean	True or False
include_waveforms	Boolean	True or False
include_testbench	Boolean	True or False
include_test_configuration	Boolean	True or False
user_comment	String	User input
logo_path	String	The path of the logo
<Template_Id>	String	Report template id
<Test_Id>	String	Executed test id
<Report_Name>	String	Report name
<report_path>[10]	string	Report path to save the report file

Returns

It generates the report of a test at a specified file location or the current working directory.

Test sequence

This section lists the functions related to test sequence.

Create a new test sequence

To create a new sequence use the following function. This function sets the response to the Clarius api instance **sequence**. You can use this instance to execute all test related functions.

To execute this function, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

[10] This argument is optional. If the path is not provided then the reports will be generated in the current working directory.

```
#Create a sequence by specifying the name and description
sequence = api.sequences.new_sequence(name, description)
#Save the sequence
id = api.sequences.save_sequence(sequence)
```

Arguments

Argument Name	Data Type	Argument Value
<name>	string	Sequence name
<description>	string	Sequence description

Note: To configure technology settings for the sequence, check *Get technology information*.

Add sequence to run a new test

To add a sequence for running a new test, use the following function.

```
new_test.add_sequence(technology, applicationName)
```

Arguments

Argument Name	Data Type	Argument Value
<technology>	string	Technology name
<applicationName>	string	Application name

Get all test sequences

To get all test sequences, use the following function.

```
sequences = api.sequences.get_all_sequences()
```

Returns

Returns the list of all test sequences.

```
[
   {
      "id":"",
```

```
        "name":"",
        "description":""
    }
]
```

Import sequence and run test

To import a saved sequence to a test and to run, use the following script. To execute these functions, use the Clarius api instance **api** from *Importing and Initializing Clarius SDK function*.

```
#Import the sequence by specifying the sequence ID
imported_sequence = api.sequences.import_sequence("Sequence ID")
#Assign the test name and test bench ID for the sequence
imported_sequence.test_name = "Test Name"
imported_sequence.testbench_id = "Test Bench ID"
#start the test
execution_id = api.tests.start_test(imported_sequence)
```

Arguments

Argument Name	Data Type	Argument Value
<Sequence ID>	string	Id of the test sequence.
<Test name>	string	Name of the test.
<Test bench id>	string	Id of the test bench.
<imported_sequence>	custom	Sequence that is created as a draft.

Modify test sequence

To modify a test sequence, use the following function.

```
#import the sequence by specifying the sequence ID
sequence = api.sequences.import_sequence("Sequence ID")
#get the technology details for the specified test category and technology
technology = sequence.get_technology(testCategory, technology)
#Set the update to be done in the application by selecting the function
technology.applications[index].set_selection(selection_status)
#update the sequence
api.sequences.update_sequence(sequence)
```

Arguments

Argument Name	Data Type	Argument Value
<Sequence ID>	string	Id of the test sequence.

Remove the sequence from the test

To remove a selected sequence from the test, use the following function. The user needs to specify the index of the application which is added to a new test.

```
new_test.remove_sequence(testCategory, technology, index)
```

Arguments

Argument Name	Data Type	Argument Value
<testCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to remove the sequence.
<technology>	string	Technology name.
<index>	int	Index of the application which starts with 0,1,2,3….

Delete a test sequence

To delete a test sequence, use the following function.

```
api.sequences.delete_sequence("Sequence ID")
```

Arguments

Argument Name	Data Type	Argument Value
<Sequence ID>	string	Id of the test sequence.

Returns

Deletes the test sequence based on the sequence id.

Others

This section describes the functions to perform tests or notifications.

Merge a test

To merge a test, use the following function.

```
test_copy = api.tests.copy_test(prev_execution_id)
technology = test_copy.get_technology("TestCategory", "TechnologyName")
technology.test_name = "new merge test"
technology.applications[index].unselect_scenario('ScenarioName')
execution_id = api.tests.merge_test(test_copy, delete_session)
```

Arguments

Argument Name	Data Type	Argument Value
<prev_execution_id>	string	Previous test execution id.
<test_copy>	custom	Copies the previous test execution request.
<Index>	int	Index of the application which starts with 0,1,2,3….
<delete_session>	boolean	True or False • True: Merges the test results • False: Test results will not be merged
<TestCategory>	Custom string. Can take only 2 values (TEST, CALIBRATION)	Test category for which you want to merge the test.
<TechnologyName>	string	Technology name.

Returns

Returns the merged test id.

Save test as a draft

To save the created test as a draft, use the following function.

```
draft_execution_id = api.tests.save_test_draft(test_draft)
```

Arguments

Argument Name	Data Type	Argument Value
<test_draft>	custom	Draft name of test.

Returns

Returns the draft id of a saved test.

Save merge test as a draft

To save the merged test as a draft, use the following function.

```
draft_execution_id = api.tests.save_merge_draft(merge_test_draft)
```

Arguments

Argument Name	Data Type	Argument Value
<merge_test_draft>	custom	Draft of a merge test.

Returns

Returns the draft id of a merged test.

Update test draft

To update the test draft, use the following function.

- Use this script to fetch a copy of the saved draft.

```
test_draft = api.tests.copy_test(draft_execution_id)
```

- Use this script to update the test draft.

```
api.tests.update_draft(test_draft)
```

Arguments

Argument Name	Data Type	Argument Value
<draft_execution_id>	string	Execution id of a test draft.
<test_draft>	custom	Draft of a test.

Returns

200: Test draft updated successfully.

400: The validation of the request failed.

Delete test draft

To delete the test draft, use the following function.

```
api.tests.delete_draft(draft_execution_id)
```

Arguments

Argument Name	Data Type	Argument Value
<draft_execution_id>	string	Execution id of a test draft.

Returns

204: Test draft deleted successfully.

401: Unauthorized error encountered. Authentication is possible but has failed or not yet been provided.

Get interrupt notifications

To get the interrupt notifications, use the following.

```
interrupt_notification = api.notifications.pull_interrupt_notifications()
```

Returns

Returns notification details.

Table 69: Response

```
[
  {
    "notificationId":"",
    "executionId":"",
    "scenarioName":"",
    "stepName":"",
    "message":"",
    "possibleActions":[
      "STOP",
      "SKIP",
      "RESUME"
    ],
    "notificationTime":"YYYY-MM-DDTHH:MM:SS"
  }
]
```

Perform interrupt action

To perform an interrupt action like STOP, SKIP, PAUSE, or RESUME, use the following.

```
from clariussdk.notifications import InterruptActions

api.notifications.perform_interrupt_action(notification_id,interrupt_action)
```

Arguments

Argument Name	Data Type	Argument Value
<notification_id>	string	id of the notification.

Table continued…

Argument Name	Data Type	Argument Value
<interrupt_action>	enum	Action that you want to perform for notification. • InterruptActions.CLEAR • InterruptActions.SKIP • InterruptActions.STOP • InterruptActions.RESUME

Returns

It displays the message as either Success or Failure.

Example

Perform interrupt action - Handle test notifications

```
interrupt_action = None
for notification in interrupt_notification:
    if "executionId" in notification and notification["executionId"] ==
execution_id:
        interrupt_action = InterruptActions.RESUME
        notification_id = notification["notificationId"]
        api.notifications.perform_interrupt_action(notification_id,
interrupt_action)
        print("Pass: Interrupt addressed successfully")
        break  # Exit the loop once the action is performed

    if interrupt_action is None:
        print("No matching executionId found in notifications.")
```

Perform interrupt action - Handle system notifications

```
from clariussdk.notifications import InterruptActions

system_notification_found = False
for notification in interrupt_notification:
    if notification.get("executionId") is None and
notification.get("possibleActions") == ["CLEAR"]:
        # Handle system notification
        print("System notification received: " + str(notification))
        notification_id = notification["notificationId"]
        api.notifications.perform_interrupt_action(notification_id,
InterruptActions.CLEAR)
        print("Pass: System notification cleared successfully")
        system_notification_found = True
```

Example script

The example script is a collection of Clarius SDK commands that are designed to be executed like a program. This often contains a set of function definitions and programs that call the APIs.

clarius

```
# This script is a sample script to demonstrate the usage of the Clarius SDK.
# This script is for representation purposes only. The actual values may vary
as per the application requirements.

from clariussdk import clarius
from clariussdk.tests import AcquisitionMode

try:
    # Initializing the SDK

    ip = input('Clarius IP: ')
    client_id = input('client id: ')
    client_secret = input('client secret: ')

    sdk = clarius.Api(ip, client_id, client_secret)

    # create a new test bench

    sample_test_bench = {
        "name": "TESTBENCH_NAME",
        "description": "",
        "technologies": [
            "TechnologyName"
        ],
        "hubAddress": "http://<<instrument service address>>:<<portid>>",
        "instruments": [
            {
                "name": "Scope",
                "type": "SIGNAL_ANALYZER",
                "category": "RT_SCOPE",
                "address": "GPIB8::1::INSTR",
                "description": "",
                "properties": {
                    "manufacturer": "TEKTRONIX",
                    "model": "default"
                },
                "extensions": []
            }
        ],
        "availability": ""
    }

    sdk.test_benches.create_testbench(sample_test_bench)
```

```
    # create 'new test'

    test_name = input('Test Name: ')
    test_bench_id = input('Test Bench ID: ')
    description = input('Description: ')

    new_test = sdk.tests.new_test(test_name, test_bench_id, description)
    new_test.acquisition_mode = AcquisitionMode.LIVE

    # add application
    app_name = 'Application Name'  # Name of the application
    technology = 'Technology' # Name of the technology. Can be fetched using
method get_available_technologies() Pg.104
    testCategory = 'Test Category' # Category of the Test.  Can take 2
values: 'TEST' or 'CALIBRATION'

    index = 0
    new_test.add_sequence(technology,app_name)
    technology = new_test.get_technology(testCategory, technology)
    technology.applications[index].set_selection(True)

    # start 'new test'
    execution_id = sdk.tests.start_test(new_test) # Actual Test Execution.
    print("execution_id:{0}".format(execution_id))

    # get 'test report'

    report_templates = sdk.templates.get_report_templates() # Get the list of
report templates.
    report_name = 'sample_report' # Name of the report
    report_path = 'path' # Path where the report needs to be saved

    sdk.reports.generate_report(execution_id,template_id =
report_templates[0],report_name = report_name,report_path=report_path) # For
additional customization options supported refer to the documentation

    # download test waveforms

    wfm_path = 'path' # Path where the waveforms needs to be saved
    sdk.results.download_waveforms(execution_id,wfm_path=wfm_path)

    except Exception as e:
    print(e)
```

Index

A

C

D

E

G

I

L

M

P

R

S

T

U

V

W