

2.5 - API Test Designer

API Test Designer — Appvance IQ documentation.

- [API Test Designer](#)
- [Service Suite API Testing Interface](#)
- [API Testing Building Blocks in Service Suite](#)
- [API Testing Examples in Service Suite](#)
- [Common JavaScript APIs](#)

API Test Designer

Contents

- [API Testing Interface](#)
- [API Testing Building Blocks](#)
- [API Testing Examples](#)
- [API Testing Authentication](#)

API (Application Programming Interface) testing is focuses on testing the interfaces or communication channels between different software components or systems.

APIs enable software applications to communicate with each other, and are used extensively in modern software development, especially in the context of web applications and cloud-based services. APIs allow developers to access data, functionality, and services provided by other systems, and to integrate their own applications with those systems. API testing is useful if there is no UI or you need to validate an API by testing the input and output values and fields.

API testing allows you to automate complex tasks with no need for Java Script Coding, and also allows the use of REST and SOAP calls.

SOAP

SOAP (Simple Object Access Protocol) is a message protocol that enables the distributed elements of an application to communicate. SOAP can be carried over a variety of standard protocols, including the web-related Hypertext Transfer Protocol (HTTP).

RESTful API

REST stands for REpresentational State Transfer. A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources.

REST API works pretty much as a website does, you make a call from a client to a server, and you get a response through HTTP.

A dedicated training course on API Testing is available as part of Appvance's training offerings. See the [Appvance Training page](#) for more information.

Last updated: August 22, 2023

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Service Suite API Testing Interface

Recommended guidance

Use the Service Suite API testing interface to build, run, and validate HTTP API checks in AIQ. The interface should help users configure requests, inspect responses, assert outcomes, and reuse extracted data.

What to review

- Request method, URL, headers, parameters, body, and authentication.
- Environment variables, DPL values, and reusable configuration.
- Response status, headers, body, timing, and extracted values.
- Assertions for technical validity and business expectations.
- Output Logs and evidence needed for troubleshooting.

Best practices

1. Start with a single known-good request before chaining calls.
2. Separate authentication/setup calls from business validation where useful.
3. Mask tokens and secrets in shared logs or screenshots.
4. Document dependencies on external APIs, rate limits, and test data.

Last updated: October 6, 2026

API Testing Building Blocks in Service Suite

API Testing Building Blocks in Service Suite

Service Suite API tests are built from reusable pieces that define the request, authentication, data inputs, validations, and follow-up actions. Keeping these blocks small and explicit makes API scenarios easier to debug and reuse.

Core building blocks

- **Request definition:** Method, URL, headers, query parameters, and body.
- **Authentication:** API keys, bearer tokens, OAuth flows, or environment-specific credentials stored securely.
- **Assertions:** Status code, response time, headers, JSON/XML fields, and business-level response values.
- **Extraction:** Capture tokens, IDs, and values from one response so later steps can reuse them.
- **Test data:** Use variables or DPLs for repeatable inputs instead of hard-coded one-off values.
- **Error handling:** Validate expected negative paths and make failures easy to diagnose from output logs.

Recommended practice

Start with a single working request, add authentication, then layer assertions and extraction. After the flow is stable, parameterize the data and add cleanup steps for records created during the test.

Last updated: October 8, 2026

API Testing Examples in Service Suite

Recommended guidance

Use API testing examples to teach repeatable Service Suite patterns: authentication, request construction, assertions, response extraction, data reuse, and error handling. Keep examples current and remove credentials or customer-specific values.

Recommended example coverage

- GET, POST, PUT/PATCH, and DELETE requests where appropriate.
- Headers, query parameters, request bodies, and content types.
- Authentication setup using safe placeholders.
- Status-code, schema, field, and business-rule assertions.
- Response extraction for downstream scenario or service steps.

Example hygiene

1. Use placeholder URLs, tokens, and account IDs.
2. Explain expected results and failure conditions.
3. Keep examples short enough to copy safely into a test project.
4. Validate examples after Service Suite or API changes.

Last updated: October 6, 2026

Common JavaScript APIs

Here are some common JS APIs. This is just a partial list. For the full list, see [Test Designer JS Commands Reference](#).

`_formSubmit(_form("formID"))`

Form Submit API which can be used instead of Enter Key if the existing page has form in it.

`_getAttribute(_byId('table_name'),"rows")`

Used to find the rows in a table and store it in a variable. The following code snippet can be used to validate whether a checkbox is checked or not.

```
var $ch = _getAttribute(_byId('accessor'),"checked");_log($ch)if($ch==false){  
_click(_checkbox("accessor")); }else{_assertEqual($ch,"checked"); }
```

`var $msg=_getText(_div("custdialog"))`

Fetches the text from a dialogue box and stores it in a variable.

- Variants to the above `getText` are as follows:
 - `getTextSelenium` - Follows the same syntax as above. The difference is it exactly trims the data without having any white spaces.

`var $ch2=eval(_lastAlert())`

Stores the contents of the last pop up into a variable which can later be compared using assertion. If a particular list has a form action, then the following code snippet can be used.

```
_setSelected(_select("accessor ID"),"list Element");_formSubmit(_form("form ID"));
```

`_mouseOver(_div("accessor"))`

Simulates Mouse Over action.

`_setFile2(_byXPath('//*[@id="fileBox"]'), "Test.mp4")`

Used for file upload actions/

- `setFile2` API is similar to `setFile` API, which uses the same syntax as that of `setFile` API, `setFile2` is used when you want to mimic a user activity where front end validations are involved.
- While using `setFile2` for file upload make sure you do not give any absolute path or relative path. Example: `_setFile2(_byXPath('//*[@id="fileBox"]'), "Test.mp4");`

`var $length = _getLength('.tablename tr')`

Jquery used to get table length. Its similar to usage in jquery format with selectors except that there we will use `$.length`. Here its not needed.

- Example Jquery notation: `$('.tableclass tr').length`
- Test Designer usage: `getLength('.tableclass tr')` (class selector)
Note: Use `#` if its an id.
- To find column count too, use : `getLength('.tableclass tr td')` or `getLength('.tableclass tr th')`

`var $ch=_getAttribute(_link("URL Name"),"href")`

Gets the HREF URL which can then be stored in a string.

`_setFile(_byXPath('//*[@id="fileBox"]'), "{ds}/Test.mp4")`

Upload file.

getComputedStyle(_div("accessor"),"background-color")

Retrieves background color or any style attribute.

rteWrite(iframe("identifier"),"Test in RTE")

Works inside rich text editor which will always be embedded inside iframe.

EVAL Usage in Test Designer JS

These can be used if in case no other option is available but to stick with pure jquery usage:

Examples:

```
_eval("ds$('html, body').scrollTop(ds$(document).height())");var data1 =  
_eval("Math.round(ds$(window).scrollTop())");log(data1);var data2 =  
_eval("ds$(window).height()");_log(data2);var data4 = +data1 + +data2 ;_log(data4);var data3 =  
_eval("ds$(document).height()");_log(data3);var end =  
_eval("Math.round(ds$(window).scrollTop()+ ds$(window).height() ==  
ds$(document).height());_log(end);if(end == true){_log("scroll bottom is reached");}
```

```
var collection = _eval("ds$('.space-program-detail').not('.hideProgram').find('.program-  
name')");log(collection.length);
```

```
_log("typeof "+typeof _eval("(function(){})());");_log( _eval("(function(){return  
null;})();");_eval("ds$('html, body').scrollTop(ds$(document).height())");
```