

3.1 - Services Workbench

Service Suite documentation.

- [Service Suite Overview](#)
- [Why Service Suite?](#)
- [Netboard Service Usage in Service Suite](#)
- [Run Node.js Packages from Service Suite](#)
- [WSDL/SOAP Call-in Service Suite](#)

Service Suite Overview

Recommended guidance

Service Suite is AIQ's API and service testing area for building HTTP requests, validating responses, extracting values, and supporting API-driven test workflows. Use "Service Suite" in current documentation instead of older Services Workbench wording unless referring to legacy navigation or historical assets.

What Service Suite supports

- HTTP API requests with headers, parameters, bodies, and authentication.
- Assertions for status codes, response fields, schemas, and business rules.
- Response extraction for reuse in later service or Scenario Editor steps.
- Data-driven API tests using DPLs or controlled test data.
- Evidence through logs, reports, and reusable service assets.

Best practices

1. Start with small, independent requests before chaining complex workflows.
2. Use secure credential handling and mask sensitive values in shared evidence.
3. Document dependencies on external services, rate limits, and environments.
4. Review older Services Workbench pages for consolidation into Service Suite content.

Last updated: October 6, 2026

Why Service Suite?

Why Service Suite?

Service Suite is used to design, run, and validate API and service-level tests in AIQ. It helps teams test business workflows below the UI, validate integrations, and reuse service calls across broader test coverage.

When Service Suite is a good fit

- Testing HTTP APIs, service integrations, and backend workflows.
- Validating data contracts, status codes, response payloads, and downstream values.
- Creating reusable service steps that support Scenario Editor or end-to-end tests.
- Running checks that should be faster or more stable than browser-only validation.
- Preparing setup, teardown, or data-creation steps for larger test suites.

Recommended practice

Use Service Suite with clear request definitions, secure authentication handling, assertions, response extraction, and environment-specific variables. Keep service tests small enough to debug quickly, then combine them into broader workflows where needed.

Last updated: October 8, 2026

Netboard Service Usage in Service Suite

Last reviewed: September 25, 2026

Use this article when a Service Suite workflow needs to call or validate the Netboard service.

What changed in this article

- Clarified that this is Service Suite integration guidance, not a general product overview.
- Added checks for endpoint availability, request payload, authentication context, and expected response shape.
- Added troubleshooting details around response validation and output logging.

Validation workflow

1. Confirm the Netboard endpoint and required request fields.
2. Run the Service Suite call with a known test payload.
3. Validate the response body, status, and any downstream values stored for later steps.

Review note: Kept the article but scoped it to Service Suite usage and validation.

Last updated: September 25, 2026

If you have feedback on this topic, contact documentation@appvance.ai.

Run Node.js Packages from Service Suite

Run Node.js Packages from Service Suite

Service Suite can invoke approved Node.js and npm-based processes on the AIQ execution host. Install Node.js, npm, and required packages on the same Test Node or host that runs the Service Suite test.

Plan for the call to wait until the external process completes. Capture and validate the process exit status, standard output, and standard error in the test logic.

Use a controlled working directory and avoid installing untrusted packages during execution. Validate package compatibility with the deployed AIQ and Node.js versions before use.

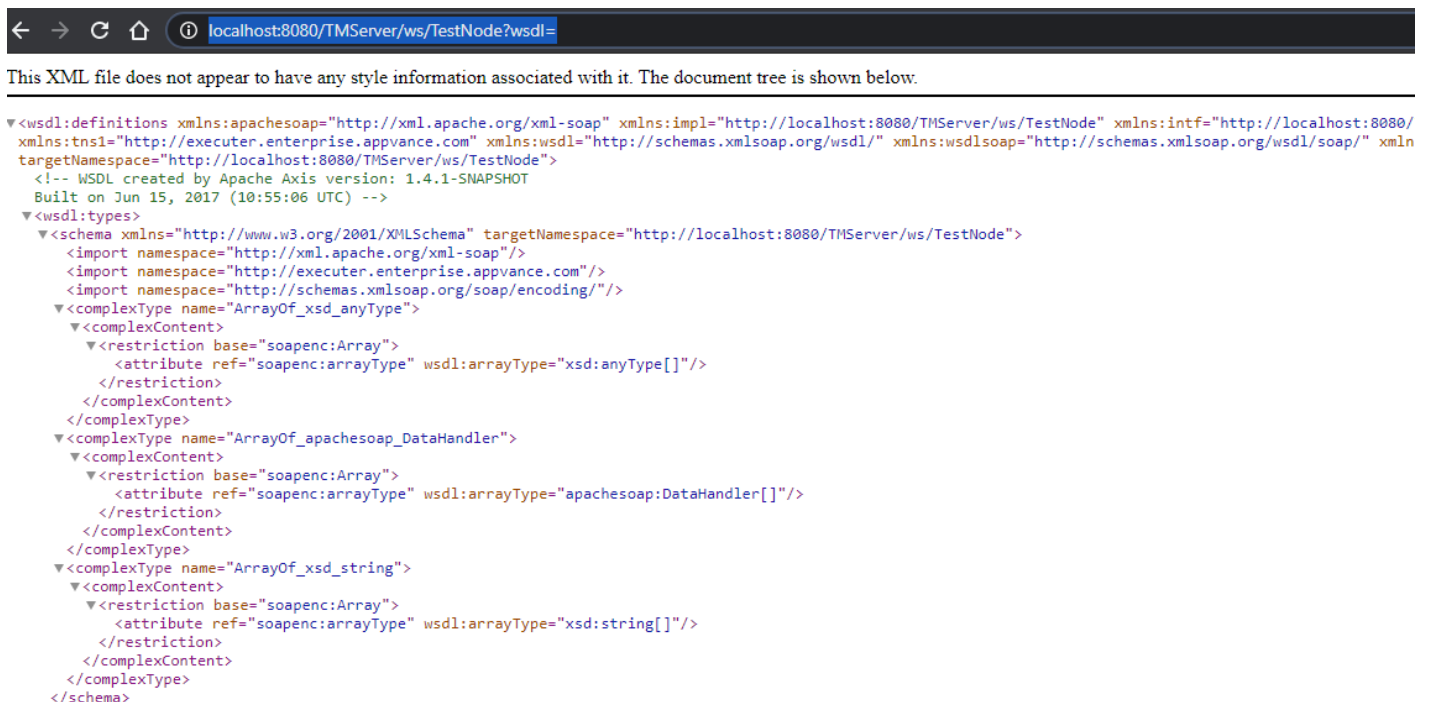
Last updated: September 14, 2026

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

WSDL/SOAP Call-in Service Suite

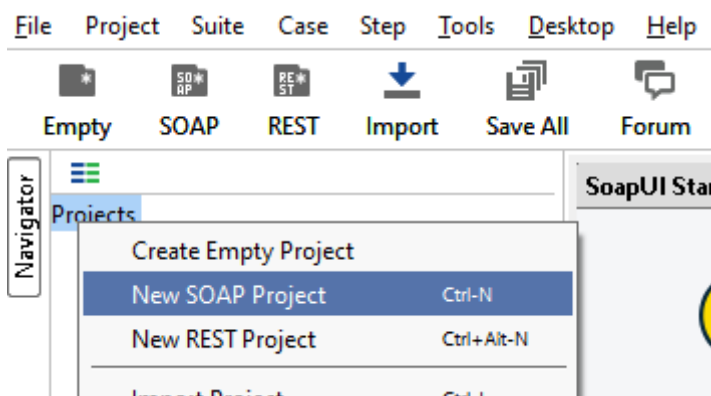
SoapUI support was removed in release 5.2.0 of AIQ.

You first need a working WSDL server, like the Appvance Server back end in <http://SERVER:8080/TMServer/ws/TestNode?wsdl=>



From there you could create manually **a post body data**, or using a third-party tools like soapUI, you could see the post's body, as we only need the body to create the post-call.

(SoapUI can be downloaded from <https://www.soapui.org/>)



A new Soap Project will show a screen to enter the URL

 New SOAP Project
 ✕

New SOAP Project


Creates a WSDL/SOAP based Project in this workspace

Project Name:

Initial WSDL:

Create Requests: ☒ Create sample requests for all operations?

Create TestSuite: ☐ Creates a TestSuite for the imported WSDL

Relative Paths: ☐ Stores all file paths in project relatively to project file (requires save)



The above will create a project with an echo method. We can open it and extract the XML section to be used as the POST body data:

```

<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema" xmlns:soapenv="
http://schemas.xmlsoap.org/soap/envelope/" xmlns:tms="http://tmserver.enterprise.appvance.com
">

<soapenv:Header/>

<soapenv:Body>

<tms:echo soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<echo xsi:type="xsd:string">?</echo>

</tms:echo>

</soapenv:Body>

</soapenv:Envelope>

```

Notice the "?", this will be our variable. We should replace "?" in the `<echo xsi:type="xsd:string">?</echo>` line with a variable name like "ECHO1" so we can later replace that for the intended parameter. For this example, we will replace "?" with "ECHO1" and save the XML as a "requests.echo.xml" file next to our "main.js".

```
//Read template
```

```
var echoTemplate = new java.lang.String(readBytes("{ds}/request.echo.xml"));
```

```
//Process template for desired method
```

```
var echoString=echoTemplate.replace("ECHO1","Hello World");
```

```
//Create post method
```

```
var echoPost = createPost("http://localhost:8080/TMServer/ws/TestNode",echoString,"text/xml");
```

```
echoPost.addHeader("SOAPAction","");
```

```
var result=process(echoPost);
```

The “SOAPAction” HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. SOAP places no restrictions on the format or specificity of the URI or that it is resolvable. An HTTP client MUST use this header field when issuing a SOAP HTTP Request. The header field value of the empty string ("") means that the intent of the SOAP message is provided by the HTTP Request-URI. No value means that there is no indication of the intent of the message. You can read more about it in https://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383528



NTLM Service Authentication

When working with services that have NTLM authentication used normally for C# services, there is some additional configuration that needs to be taken into consideration. One of the requirements to consume this type of service will be to provide the credentials configured in the IIS for the available users to log in to the server. The system administrator will provide the domain and workstation where the services reside, similarly, it should provide the username and the password. Once you have the credentials information you will need to proceed to introduce this in the script which has the service call as the following example;

appvance IQ

Dashboard

Test Designer

Mobile Designer

AI Scripting

Service Suite

New

Scenario Editor

Play

Analyze

DPL Options

Traffic

File

serviceSuiteTestNTLM.js

Play Script

```
1 log('hi test')
2 var $url = "https://applicationurl/MembershipService.svc";
3 var proxyPort;
4 try {
5   setNTLMCreds({
6     "password": "TRI-test01",
7     "domain": "tri-technical.domain.com",
8     "workstation": "p2atrimis.domain.com",
9     "enabled": true,
10    "username": "dvdestuser14"
11  });
12  //proxyPort=startTestDesignerProxy("localhost",9090);
13  //var $postData = "";
14  var $postData = new java.lang.String(readBytes("{ds}/xmlBody.xml"));
15  log($postData);
16  //setProxy("localhost",9094, "Avttestuser14", "TRI-test01");
17  var postCall = createPost($url, $postData,"text/xml");
18  //postCall.addHeader("WWW-Authenticate","NTLM");
19  postCall.addHeader("SOAPAction","http://tempuri.org/service/GetAllBrands");
20  var resp = process(postCall);
21  log(resp.toStringContent())
22 } finally {
23   //stopTestDesignerProxy("localhost",9090,proxyPort);
24 }
```

Last updated: July 25, 2024

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