

Using Postman to test SOAP web services

Learn how to use Postman to create automated API tests for SOAP web services



Oluwatomisin Bamimore

Published September 29, 2022



Photo by 'Nathan Powers' on Unsplash

TABLE OF CONTENTS



AI Assistant for Playwright

Code AI-powered test steps with the free ZeroStep JavaScript library

[Learn more](#)

Nowadays most server-to-server communication over HTTP is done via REST APIs that use JSON as its serialization format. However, if you're working with applications that integrate with any legacy systems, you may find yourself needing to communicate via the SOAP messaging standard.

In this article, we'll walk through how to test SOAP services using a popular API testing tool called [Postman](#). But first, let's cover some of the key things to know about SOAP.

What is SOAP?

SOAP, which stands for Simple Object Access Protocol, is an XML-based messaging standard for handling structured data that was created by Microsoft in 1998.

A typical SOAP message consists of four blocks:

1. Envelope: The entirety of a SOAP message is wrapped in the `<Envelope>` block.
2. Header: This is an optional section inside the `<Envelope>` which contains information that should be processed before the `<Body>` section. Authentication credentials are one example of what may be defined within the `<Header>` section.
3. Body: The body contains the actual payload of the SOAP message. For example, if we're using SOAP to issue remote procedure calls (RPCs), we would likely use the `<Body>` section to define the RPC call that we're attempting to make, along with any parameters that should be sent along.
4. Fault: The `<Fault>` block contains information on any errors that occurred while processing a SOAP message.

SOAP WSDL

Whereas with REST APIs you might use something like the [OpenAPI specification](#) to describe your API, SOAP shipped with its own definition language called WSDL (commonly pronounced “whizz-dull”). WSDL stands for Web Service Description Language, and it’s a machine-readable way to define what is supported in your SOAP API.

Here’s an example WSDL that defines a simple stock quote service:

```
1  <?xml version="1.0"?>
2  <definitions name="StockQuote"
3      targetNamespace="http://example.com/stockquote.wsdl"
4      xmlns:tns="http://example.com/stockquote.wsdl"
5      xmlns:xsd1="http://example.com/stockquote.xsd"
6      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7      xmlns="http://schemas.xmlsoap.org/wsdl/">
8
9  <types>
10 <schema targetNamespace="http://example.com/stockquote.xsd"
11     xmlns="http://www.w3.org/2000/10/XMLSchema">
12     <element name="TradePriceRequest">
13         <complexType>
14             <all>
15                 <element name="tickerSymbol" type="string"/>
16             </all>
17         </complexType>
18     <element name="TradePrice">
19         <complexType>
20             <all>
21                 <element name="price" type="float"/>
22             </all>
23         </complexType>
24     </element>
25 </schema>
26 </types>
27 <message name="GetLastTradePriceInput">
28     <part name="body" element="xsd1:TradePriceRequest"/>
29 </message>
30 <message name="GetLastTradePriceOutput">
31     <part name="body" element="xsd1:TradePrice"/>
32 </message>
33 <portType name="StockQuotePortType">
34     <operation name="GetLastTradePrice">
35         <input message="tns:GetLastTradePriceInput"/>
36         <output message="tns:GetLastTradePriceOutput"/>
37     </operation>
38 </portType>
39 <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
40     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http">
41         <operation name="GetLastTradePrice">
42             <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
43             <input>
44                 <soap:body use="literal"/>
```

```
45      </input>
46      <output>
47          <soap:body use="literal"/>
48      </output>
49  </operation>
50 </binding>
51 <service name="StockQuoteService">
52     <documentation>My first service</documentation>
53     <port name="StockQuotePort" binding="tns:StockQuoteSoapBind:
54         <soap:address location="http://example.com/stockquote"/>
55     </port>
56 </service>
57 </definitions>
```

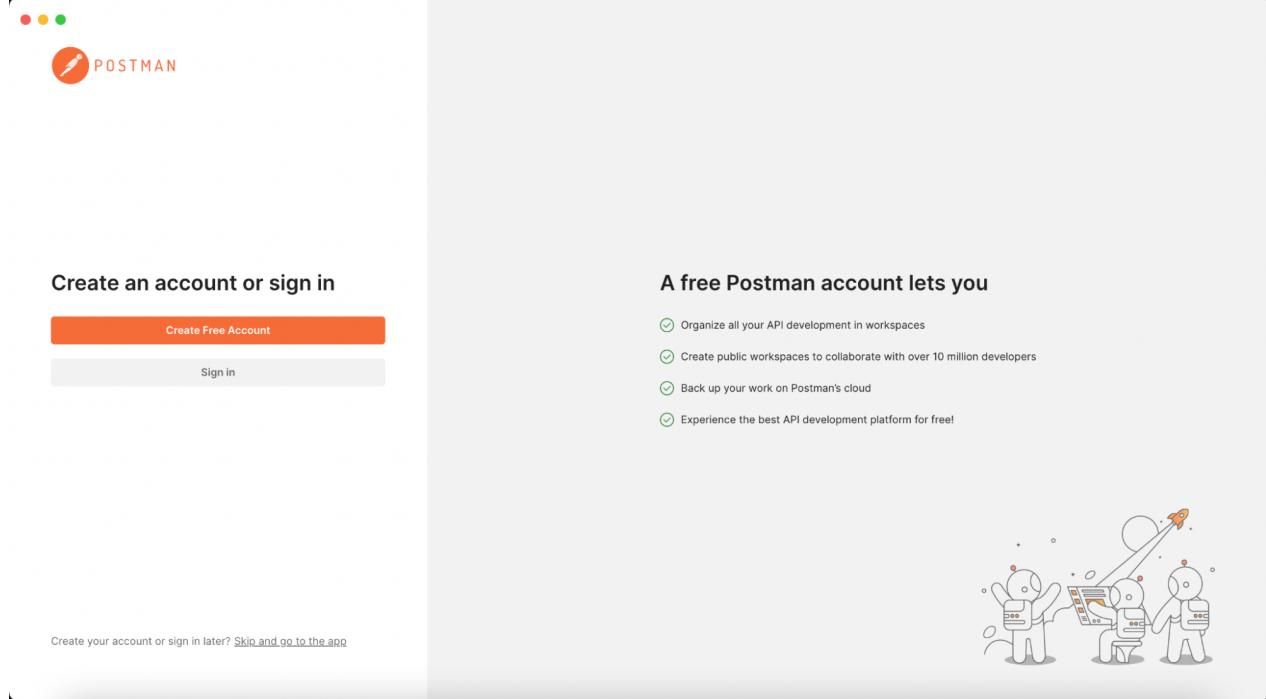
Source credit: <https://cs.au.dk/~amoeller/>

Testing SOAP APIs in Postman

If you're building or consuming SOAP services, you should be thinking about how to efficiently write tests against it. Postman is a popular application for building, testing, and maintaining APIs that was created in 2012 by Abhinav Asthana, and is what we'll be using to test out a sample SOAP service. We'll briefly walk through how to install Postman, and then discuss how to use it to interact with a SOAP service.

Installing Postman

Download and install Postman at <https://www.postman.com/downloads/>. If the installation is successful, you should see the home screen when you launch Postman on your computer.



Create a free account to complete the installation.

Note that Postman also has a [web interface](#), so if you prefer, you can follow the examples in this article without installing the standalone desktop app.

The SOAP API we will be testing against is a “Text Casing Service”. This SOAP service, which is publicly available, is pretty simple: it changes the capitalization of the words sent to it.

You can find the WSDL document for the Text Casing Service at
<https://www.dataaccess.com/webservicesserver/TextCasing.wso?WSDL>.

Creating Testcases With Postman

One of the options in the API is `InvertStringCase`. For example, sending “AbCdE” to the endpoint should get “aBcDe” as a response.

Use the following steps to test the `InvertStringCase` feature.

Step 1: Add a new blank request in Postman

On the main screen of Postman, click the ‘+’ button to create a new blank request.

My Workspace

Overview

My Workspace

Add a brief summary about this workspace

This is your personal, private workspace to play around in. Only you can see the collections and APIs you create here - unless you share them with your team.

Activity

Filter by Elements People

In this workspace

- Requests
- Collections (0)
- APIs (0)
- Environments (0)
- Mock Servers (0)
- Monitors (0)
- Flows (0)

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create collection

Online Find and Replace Console

Cookies Capture requests Bootcamp Runner Trash

Step 2: Define the HTTP method and URL

Select the POST HTTP request method and set the request URL to
<https://www.dataaccess.com/webserviceserver/TextCasing.wso>

POST https://www.dataaccess.com/webserviceserver/TextCasing.wso

POST Headers (10) Body Pre-request Script Tests Settings

Body Type: raw XML

Body Cookies Headers (8) Test Results (1/1)

Status: 200 OK Time: 77 ms Size: 549 B Save Response

Online Find and Replace Console

Step 3: Add the request body

In order to interact with the service, we'll need to send a SOAP message that validates against the TextCasing WSDL that we linked to above. You can add the following XML request body by selecting "raw" and "XML" as the text type:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
3      <soap12:Body>
4          <InvertStringCase xmlns="http://www.dataaccess.com/webserviceserver/">
5              <sAString>AbCdE</sAString>
6          </InvertStringCase>
7      </soap12:Body>
8  </soap12:Envelope>

```

The XML body was constructed using the specification in the APIs WSDL file. The `<sAString>` element is critical because it contains the actual string we want to get inverted.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Your collection'. The main area shows a POST request to 'https://www.dataaccess.com/webserviceserver/TextCasing.wso'. In the 'Body' tab, the XML code is pasted into the raw input field. The XML code is identical to the one shown in the code block above, with the `<sAString>` element highlighted by a red box.

Step 4: Set the Content-Type header

Next, we'll set the Content-Type header to `text/xml`. This step is necessary because the Content-Type header tells the SOAP server to treat the request body as XML and not any other media type.

Click on "Headers":

The screenshot shows the Postman interface with a 'POST' request to <https://www.dataaccess.com/webserviceserver/TextCasing.wso>. The 'Headers' tab is selected, showing 10 headers. The 'Body' tab is also visible. The 'Body' section contains the following XML code:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
3      <soap12:Body>
4          <InvertStringCase xmlns="http://www.dataaccess.com/webserviceserver/">
5              <sAString>AbCdE</sAString>
6          </InvertStringCase>
7      </soap12:Body>
8  </soap12:Envelope>

```

At the bottom, the status is 200 OK, time is 244 ms, and size is 549 B.

Then set the Content-Type header to text/xml and check the associated checkbox:

The screenshot shows the Postman interface with the same request setup. In the 'Headers' table, the 'Content-Type' row is selected and highlighted with a purple border. The 'Value' column for 'Content-Type' is set to 'text/xml'. The 'Key' column is 'Content-Type' and the 'Value' column is 'text/xml'.

Step 5: Submit your request

Now we're ready to submit our SOAP request! Hit the "Send" button, and the SOAP service should return an XML response that has inverted the case of your input string:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">

```

```

3   <soap:Body>
4     <m:InvertStringCaseResponse xmlns:m="http://www.dataacces
5       <m:InvertStringCaseResult>aBcDe</m:InvertStringCaseRe
6     </m:InvertStringCaseResponse>
7   </soap:Body>
8 </soap:Envelope>

```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A 'Create a collection for your requests' section is also present. The main workspace has a tab for 'Your collection' with an 'Authorization' section set to 'API Key'. The URL bar shows 'https://www.dataaccess.com/webservicesserver/TextCasing.wso'. A POST request is being made to this URL. The 'Body' tab is selected, showing the XML code for the SOAP message. The response body is displayed below, also in XML format, with the output 'aBcDe' highlighted. The status bar at the bottom indicates 'Status: 200 OK'.

Step 6: Setting up test automation

Since Postman lets you save API calls, you can refer back to this API in the future and simply use the Send button again to invoke the API. However Postman also supports creating automated JavaScript-based tests that let you not only invoke the service, but also add various assertions about what data is returned.

Switch to the 'Test' tab to add some JavaScript code that gets executed **after** the request to the SOAP API.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Your collector' and 'Your collection'. The main area shows a POST request to 'https://www.dataaccess.com/webserviceserver/TextCasing.wsdl'. The 'Body' tab is active, displaying XML code. The 'Tests' tab in the top navigation bar is highlighted with a red arrow pointing to its corresponding section in the bottom right, which shows the response body in XML format.

Add the following JavaScript snippet to the console:

```
1 pm.test("InvertStringCase: AbCdE gets inverted to aBcDe", () =>
2   // Convert XML to JSON
3
4   var jsonObject = xml2Json(responseBody);
5
6   // Move through the JSON tree to find the object that contains the result
7
8   var numberToWord = jsonObject["soap:Envelope"]["soap:Body"]["InvertStringCaseResponse"]["InvertStringCaseResult"];
9
10  // Return a boolean where True means the test passed and False means it failed
11
12  return numberToWord == "aBcDe";
13);
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'APIs', 'Environments', 'Mock Servers', 'Flows', and 'History'. The main area shows a request to 'https://www.dataaccess.com/webserviceserver/TextCasing.wso' with a 'POST' method. The 'Tests' tab is selected, displaying a JavaScript test script:

```
1 pm.test("InvertStringCase: AbCdE gets inverted to aBcDe", () => {
2   // Convert XML to JSON
3   var jsonObject = xml2json(responseBody);
4   // Move through the JSON tree to find the object that contains the inverted cased string
5   var numberToWord = jsonObject['soap:Envelope']['soap:Body'][0]['m:InvertStringCaseResponse']
6   // Return a boolean where True means the test passed and False means the test failed
7   return numberToWord == "aBcDe"
8});
```

The 'Body' tab shows the XML response from the API call:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3   <soap:Body>
4     <m:InvertStringCaseResponse xmlns:m="http://www.dataaccess.com/webserviceserver/">
5       <m:InvertStringCaseResult>aBcDe</m:InvertStringCaseResult>
6     </m:InvertStringCaseResponse>
7   </soap:Body>
8 </soap:Envelope>
```

The `pm` object automatically exists in the Postman environment. It contains different methods to test API responses. `pm.test` expects a `TestName` and `SpecFunction`. The `SpecFunction` passed as an argument to `pm.test` must return a boolean `True` or `False` that indicates whether the test has passed or failed.

Postman also sets a `responseBody` variable by default. `responseBody` contains the response body from the current API call. Since the SOAP service response is in XML format, we use the `xml2Json` package to convert the response to a JavaScript object. Thankfully, Postman makes the `xml2Json` package available by default.

Step 7: Run the script

To run the test script, click the "Send" button and switch to the "Test Results" tab to see the results of the test case:

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, API Network, and Explore. The main area shows a collection named 'Your collection' with an 'Authorization' step. A red arrow points from the text in the 'Tests' tab to the 'Test Results' section below. The 'Test Results' section shows one test named 'InvertStringCase: AbCdE gets inverted to aBcDe' with a status of 'PASS'. The status bar at the bottom indicates a 'Status: 200 OK' response.

It passed, which means we have confirmed that the SOAP service inverts the case correctly!

Conclusion

Apart from ascertaining that an API returns the correct values, test cases also serve as a layer of technical documentation for APIs. The type of tests seen in this article would be consider **integration tests**, and along with unit tests and end-to-end testings, ensure that the system as a whole is working as expected.

Get started with Reflect today

Create your first test in 2 minutes, no installation or setup required. Accelerate your testing efforts with fast and maintainable test suites without writing a line of code.

[Try Reflect for free](#)

[Schedule a demo →](#)

Copyright © Reflect Software Inc. All Rights Reserved.