How-tos & Guides 6 min read

Using Postman to test SOAP web services

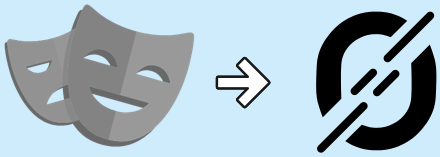
Learn how to use Postman to create automated API tests for SOAP web services

Oluwatomisin Bamimore

Published September 29, 2022

Photo by 'Nathan Powers' on Unsplash

TABLE OF CONTENTS



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:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
         <complexType>
           <all>
             <element name="price" type="float"/>
           </all>
         </complexType>
      </element>
    </schema>
  </types>
  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePor
    <soap:binding style="document" transport="http://schemas.xml
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTra
      <input>
        <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>
```

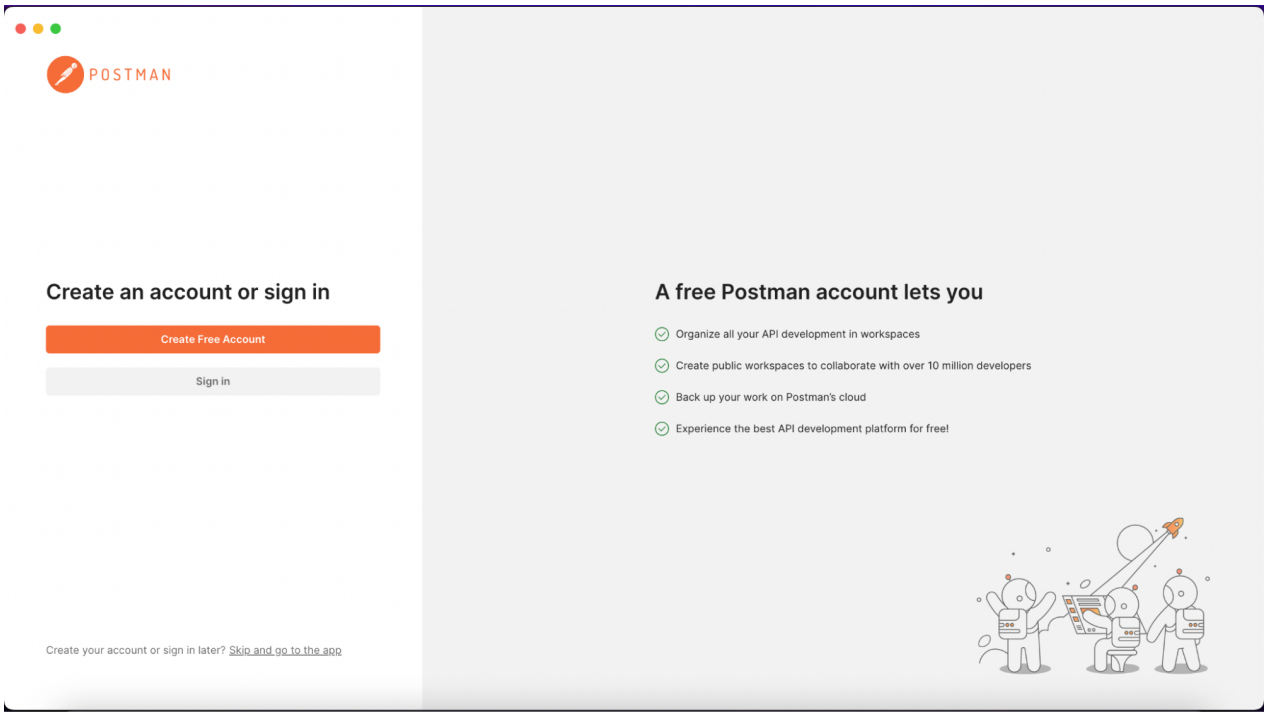
Source credit: **https://cs.au.dk/~amoeller/**

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**.

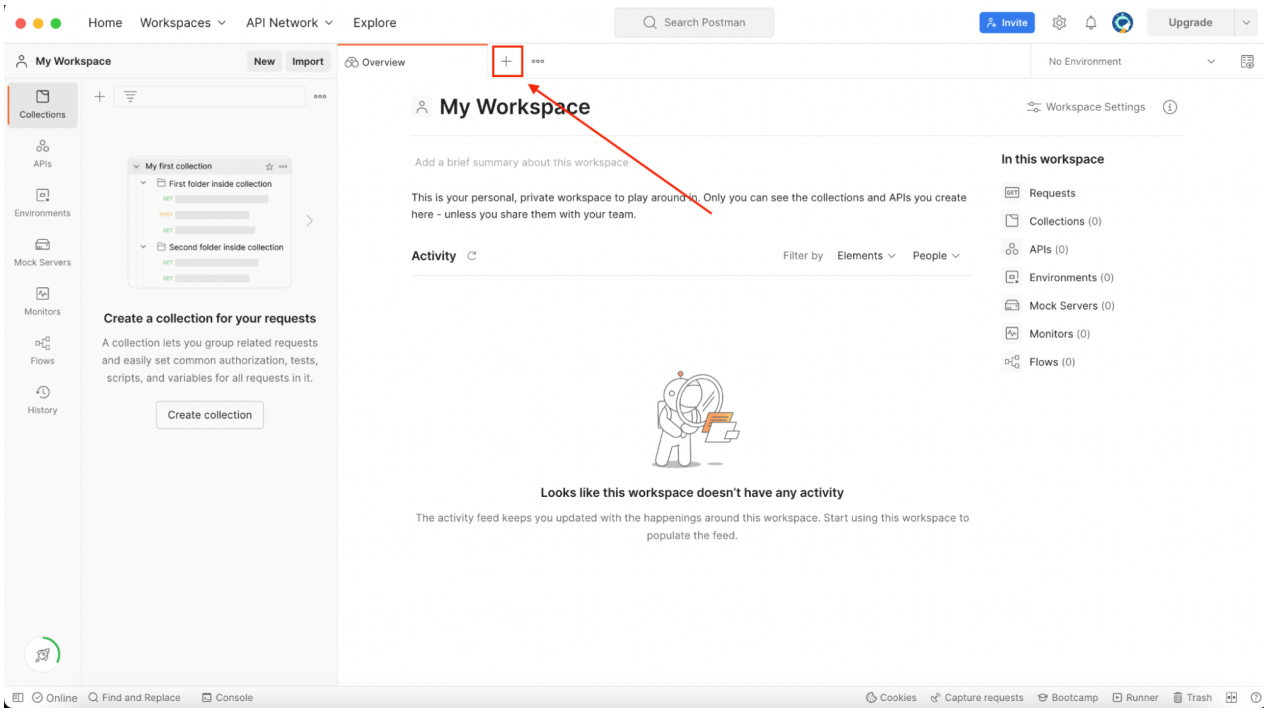
Creating Testcases With Postman

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.

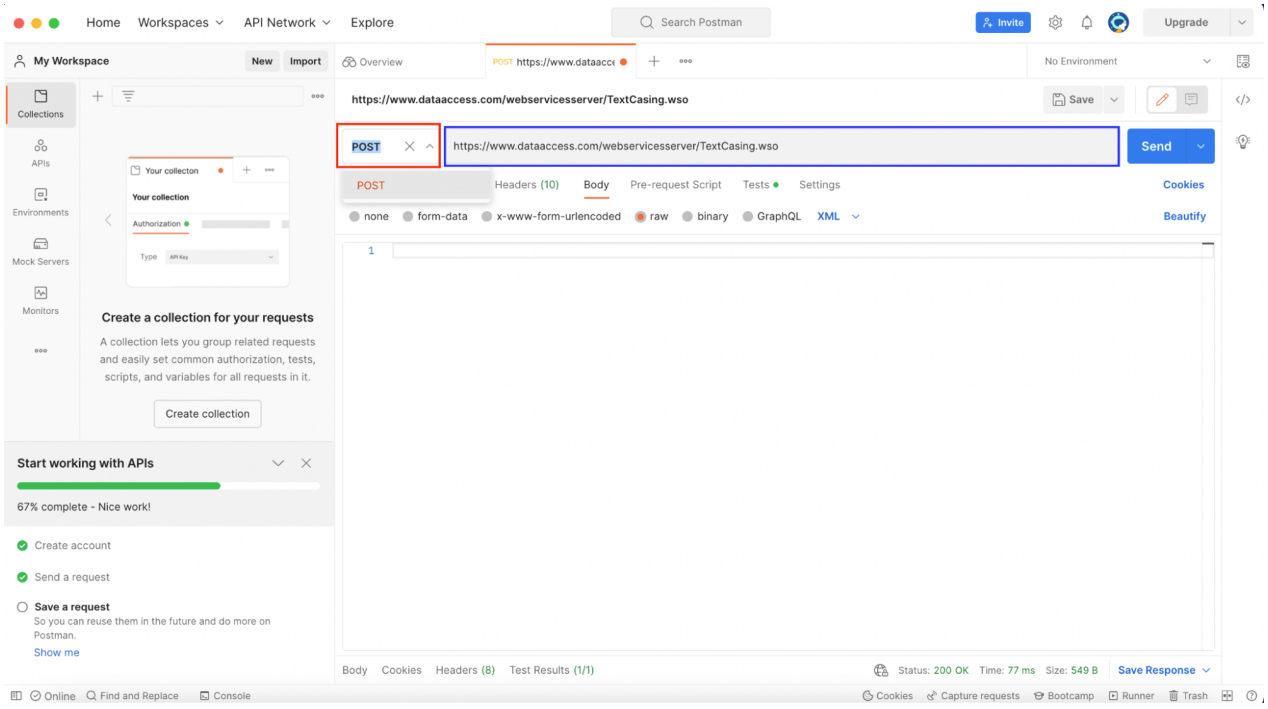
Step 1: Add a new blank request in Postman

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



Step 2: Define the HTTP method and URL

Select the `POST` HTTP request method and set the request URL to **https://www.dataaccess.com/webservicesserver/TextCasing.wso**

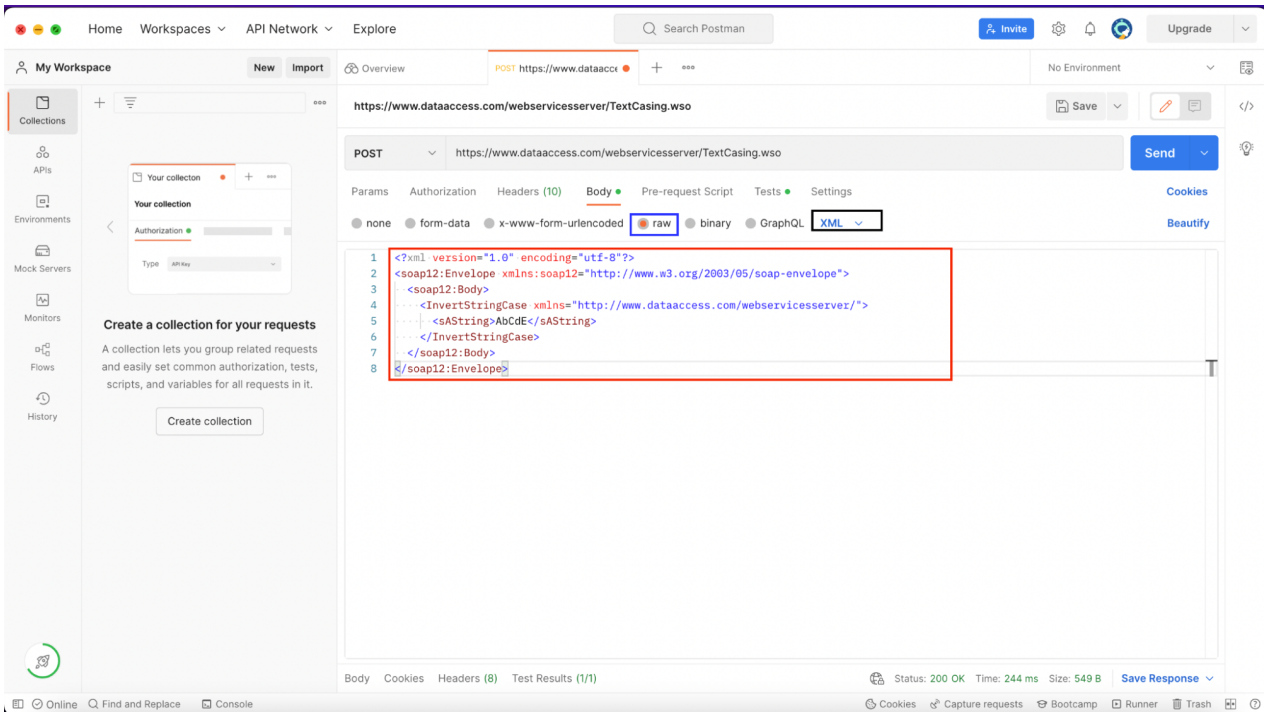


Step 3: Add the request body

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:

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-en
  <soap12:Body>
    <InvertStringCase xmlns="http://www.dataaccess.com/webservic
      <sAString>AbCdE</sAString>
    </InvertStringCase>
  </soap12:Body>
</soap12:Envelope>
```

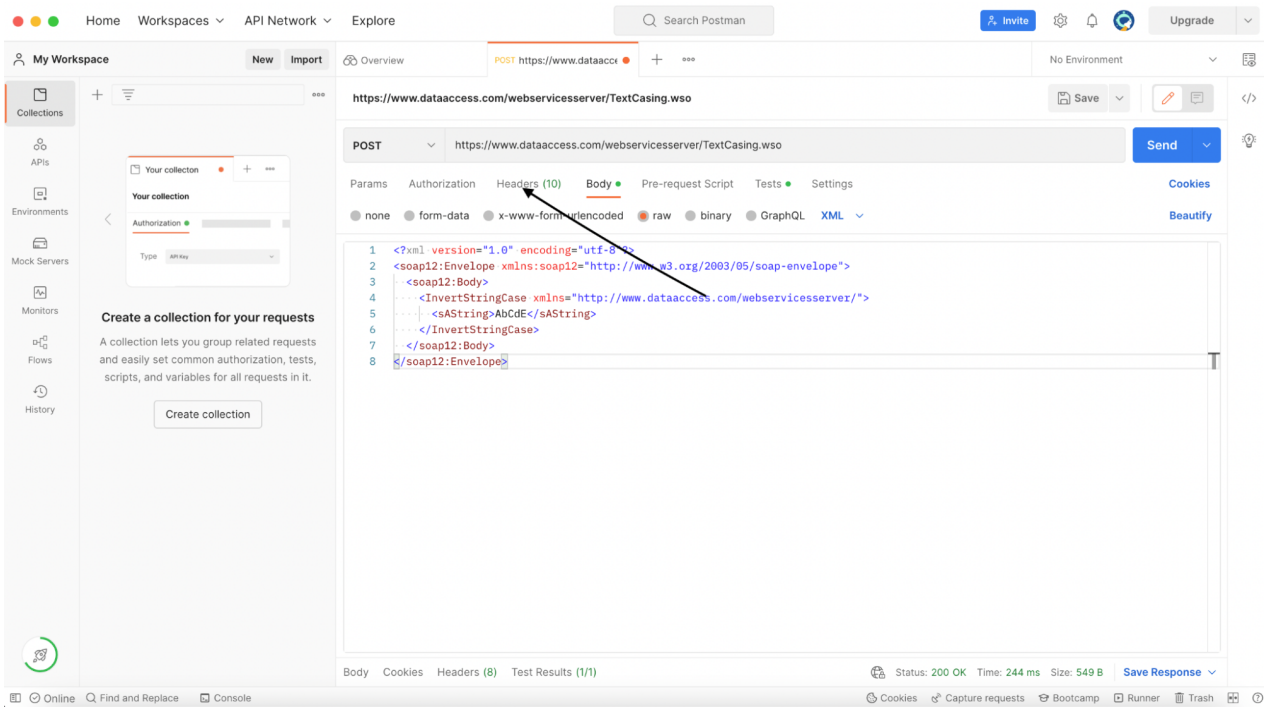
The XML body was constructed using the specification in the APIs WSDL file. The `<sAString>` element is critical because it contains the actual string we want to get inverted.



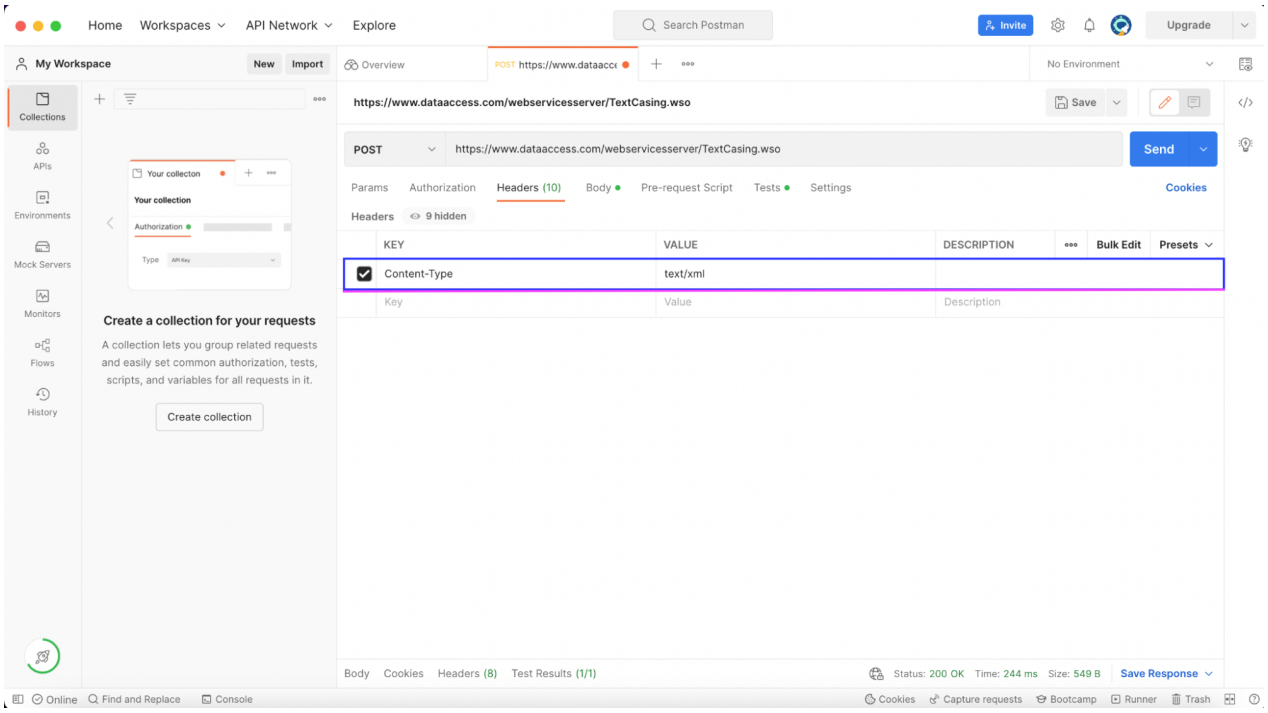
Step 4: Set the Content-Type header

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":



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

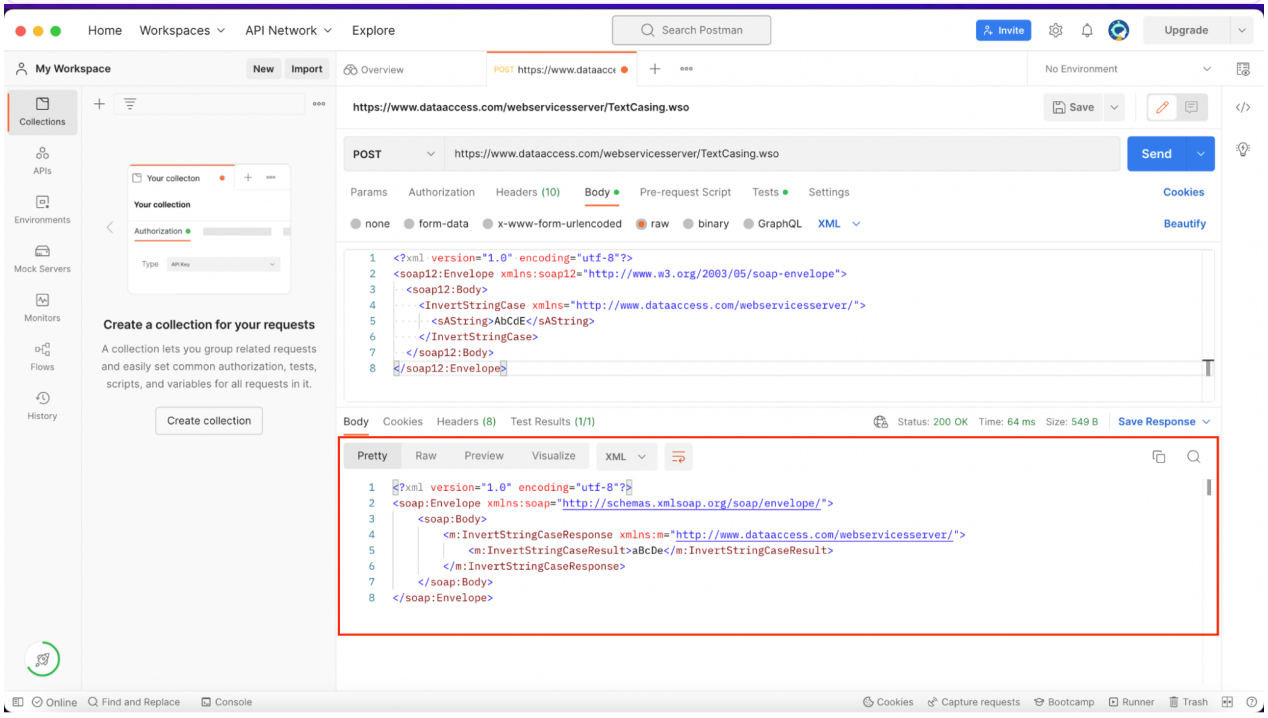


Step 5: Submit your request

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:

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

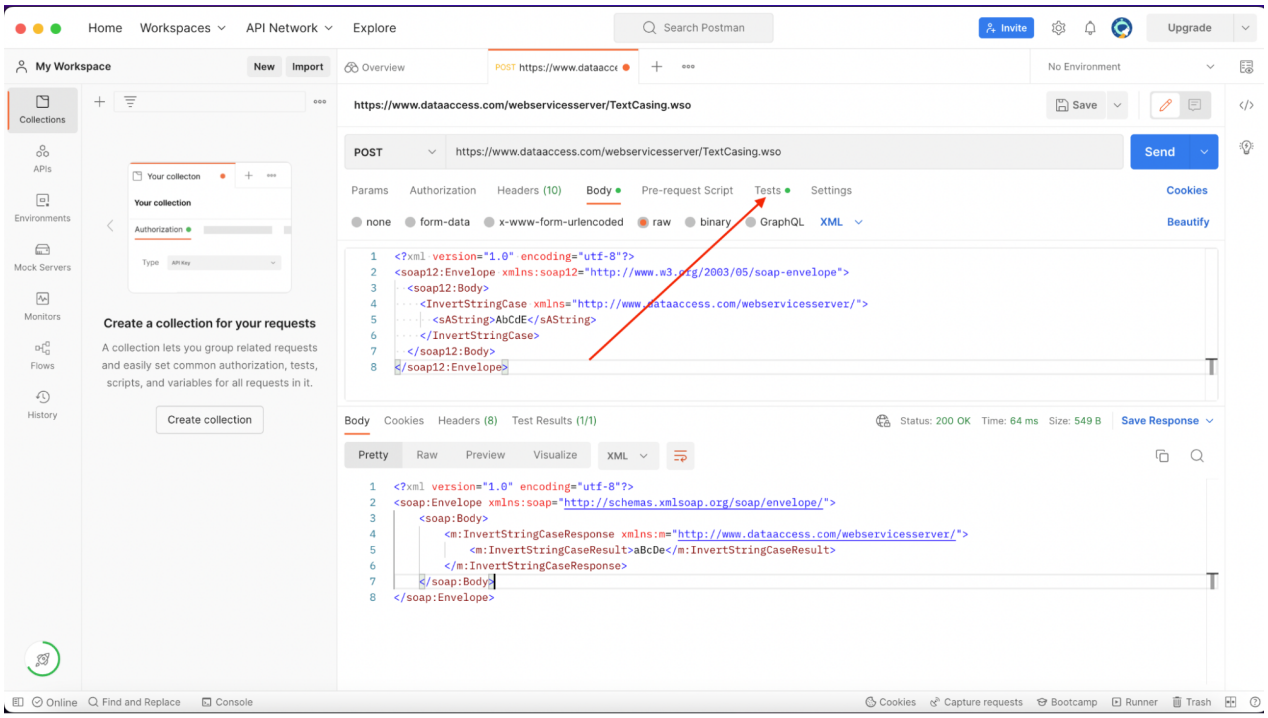
```
        <soap:Body>
            <m:InvertStringCaseResponse xmlns:m="http://www.dataacce
                <m:InvertStringCaseResult>aBcDe</m:InvertStringCaseR
            </m:InvertStringCaseResponse>
        </soap:Body>
    </soap:Envelope>
```



Step 6: Setting up test automation

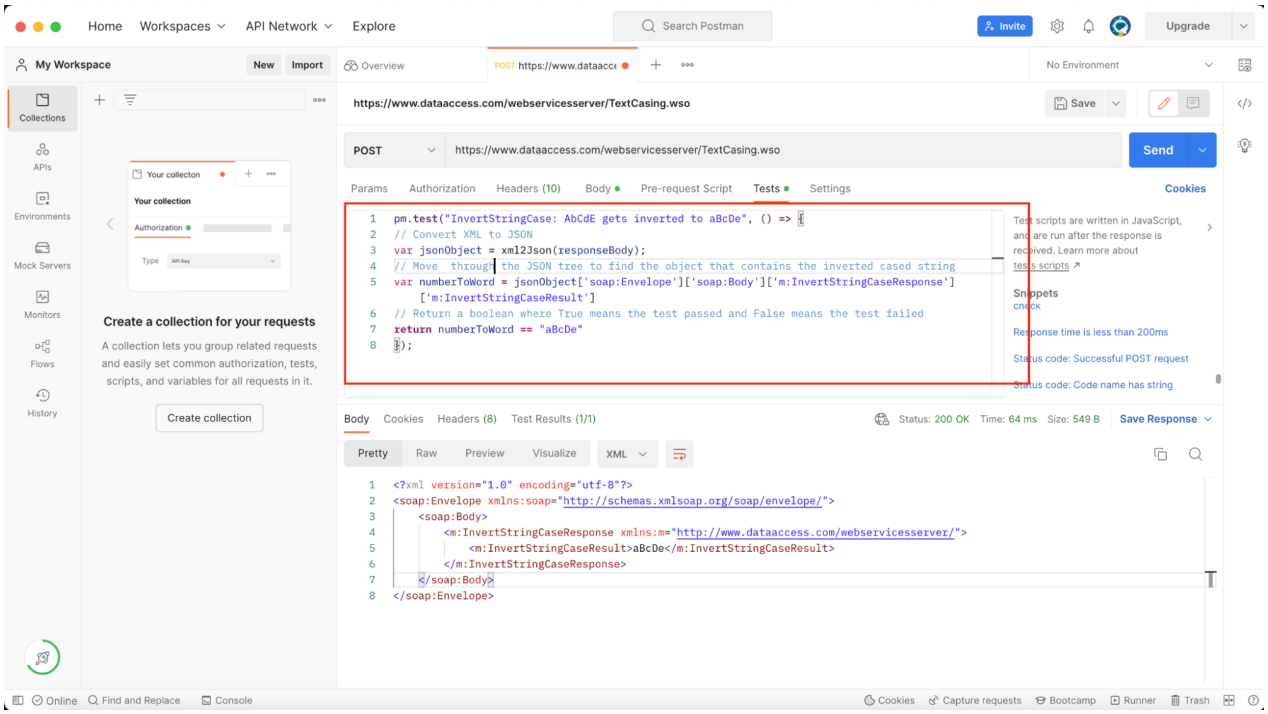
Since Postman lets you save API calls, you can refer back to this API in the future and simply use the Send button again to invoke the API. However Postman also supports creating automated JavaScript-based tests that let you not only invoke the service, but also add various assertions about what data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

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



Add the following JavaScript snippet to the console:

```
pm.test("InvertStringCase: AbCdE gets inverted to aBcDe", () =>
  // Convert XML to JSON

  var jsonObject = xml2Json(responseBody);

  // Move through the JSON tree to find the object that contains

  var numberToWord = jsonObject["soap:Envelope"]["soap:Body"]["

  // Return a boolean where True means the test passed and Fals

  return numberToWord == "aBcDe";
});
```

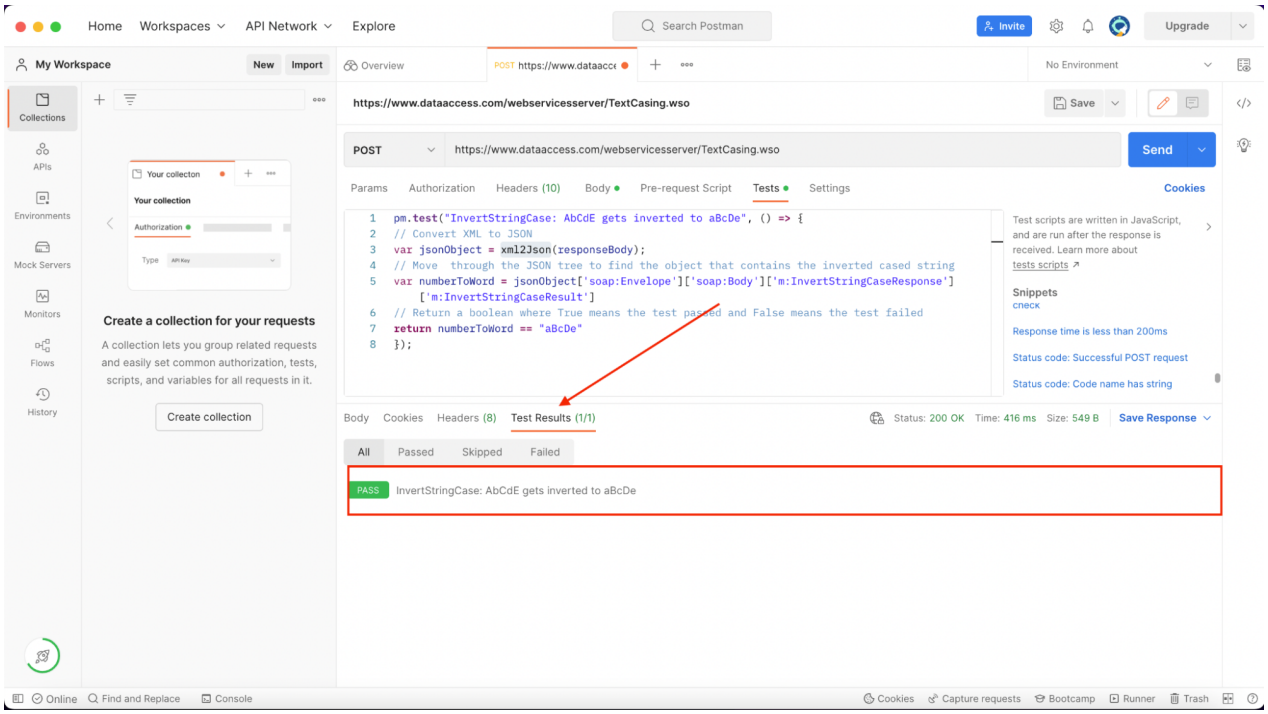


The `pm` object automatically exists in the Postman environment. It contains different methods to test API responses. `pm.test` expects a `TestName` and `SpecFunction.` The `SpecFunction` passed as an argument to `pm.test` must return a boolean `True` or `False` that indicates whether the test has passed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



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.

Schedule a demo →