

Q How can we help?

Clear

API

[Working with data](#)

[The Golden Age of APIs](#)

[SOAP vs REST APIs](#)

[REST Request Methods](#)

[Understanding REST Headers and Parameters](#)

[5 Best Practices for Data Driven API Testing](#)

[Quick REST Reference](#)

[Data Driven Testing Obstacles](#)

[How IoT Pushes QA to the Left](#)

[What Is A REST API?](#)

[Functional Testing](#)

[Load Testing](#)

[Mocking & Virtualization](#)

[Automation](#)

[Security](#)

[Open Source](#)

[Home](#) / [Learn](#) / [API](#) / SOAP vs REST APIs

SOAP vs REST 101: Understand The Differences

The age old question: *what is the difference between SOAP and REST APIs, and which one is right for my project?*

Just because our name is SoapUI, doesn't mean that we also don't know what we are talking about when it comes to explaining RESTful web services and APIs.

So, if you're looking for a resource that provides you with an answer to this age old question, you've come to the right place. We will also go over example code, as well as challenges and critiques of each choice.

We suggest starting with the video as an introduction to this topic, or for those who are just visual learners.

Start here:

Understand the Difference Between SOAP and REST APIs



The term **web API** generally refers to both sides of computer systems communicating over a network: the API services offered by a server, as well as the API offered by the client such as a web browser.

The server-side portion of the web API is a programmatic interface to a defined request-response message system, and is typically referred to as the **Web Service**. There are several design models for web services, but the two most dominant are **SOAP** and **REST**.

SOAP provides the following advantages when compared to REST:

- Language, platform, and transport independent (REST requires use of HTTP)
- Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
- Standardized
- Provides significant pre-build extensibility in the form of the WS* standards
- Built-in error handling
- Automation when used with certain language products

REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

- Uses easy to understand standards like swagger and OpenAPI Specification 3.0
- Smaller learning curve
- Efficient (SOAP uses XML for all messages, REST mostly uses smaller message formats like JSON)
- Fast (no extensive processing required)
- Closer to other Web technologies in design philosophy

As one REST API tutorial put it: SOAP is like an envelope while REST is just a postcard.

Certainly a postcard is faster and cheaper to send than an envelope, but it could still be wrapped within something else, even an envelope.

You can just read a postcard too, while an envelope takes a few extra steps, like opening or unwrapping to access what's inside.

This is just the TLDR version, keep reading below to go into more details about the two formats. Or, check out the [SOAP vs REST infographic](#) if that's more your style.



SOAP

SOAP – Simple Object Access Protocol – is probably the better known of the two models.

SOAP relies heavily on XML, and together with schemas, defines a very strongly typed messaging framework.



Every operation the service provides is explicitly defined, along with the XML structure of the request and response for that operation.

Each input parameter is similarly defined and bound to a type: for example an integer, a string, or some other complex object.

All of this is codified in the WSDL – Web Service Description (or Definition, in later versions) Language. The WSDL is often explained as a contract between the provider and the consumer of the service. In programming terms the WSDL can be thought of as a method signature for the web service.

Track Test Performance As You Scale Your API Testing

[Compare: All SoapUI Pro Features](#) →

 <h3>SoapUI Open Source</h3> <ul style="list-style-type: none"> ✓ Support for SOAP and REST API Testing. ✗ Easy multi-environment switching. ✗ Detailed test history and test comparison reporting. 	 <h3>SoapUI Pro</h3> <ul style="list-style-type: none"> ✓ Support for SOAP, REST, and GraphQL API Testing. ✓ Easy multi-environment switching. ✓ Detailed test history and test comparison reporting.
---	---

[Try SoapUI Pro](#)

Example:

A sample message exchange looks like the following.

A request from the client:

```
POST http://www.stgregorioschurchdc.org/cgi/websvccal.cgi HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml; charset=UTF-8
SOAPAction: "http://www.stgregorioschurchdc.org/Calendar#easter_date"
Content-Length: 479
Host: www.stgregorioschurchdc.org
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
<?xml version="1.0"?>
<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:cal="http://www.stgregorioschurchdc.org/Calendar">
<soapenv:Header/>
<soapenv:Body>
  <cal:easter_date soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <year xsi:type="xsd:short">2014</year>
  </cal:easter_date>
</soapenv:Body>
</soapenv:Envelope>
```

The response from the service:

```
HTTP/1.1 200 OK
Date: Fri, 22 Nov 2013 21:09:44 GMT
Server: Apache/2.0.52 (Red Hat)
SOAPServer: SOAP::Lite/Perl/0.52
Content-Length: 566
Connection: close
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <namesp1:easter_dateResponse
xmlns:namesp1="http://www.stgregorioschurchdc.org/Calendar">
<s-gensym3 xsi:type="xsd:string">2014/04/20</s-gensym3>
</namesp1:easter_dateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From this example we can see the message was sent over HTTP. SOAP is actually agnostic of the underlying transport protocol and can be sent over almost any protocol such as HTTP, SMTP, TCP, or JMS.

As was already mentioned, the SOAP message itself must be XML-formatted. As is normal for any XML document, there must be one root element: the Envelope in this case.

This contains two required elements: the Header and the Body. The rest of the elements in this message are described by the WSDL.

The accompanying WSDL that defines the above service looks like this (the details are not important, but the entire document is shown here for completeness):

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
name="Calendar" targetNamespace="http://www.stgregorioschurchdc.org/Calendar">
<message name="EasterDate">
  <part name="year" type="xsd:short"/>
</message>
<message name="EasterDateResponse">
  <part name="date" type="xsd:string"/>
</message>
<portType name="EasterDateSoapPort">
  <operation name="easter_date" parameterOrder="year">
    <input message="tns:EasterDate"/>
    <output message="tns:EasterDateResponse"/>
  </operation>
</portType>
<binding name="EasterDateSoapBinding" type="tns:EasterDateSoapPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="easter_date">
    <soap:operation soapAction="http://www.stgregorioschurchdc.org/Calendar#easter_date"/>
    <input>
      <soap:body use="encoded" namespace="http://www.stgregorioschurchdc.org/Calendar" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="http://www.stgregorioschurchdc.org/Calendar" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
<service name="Calendar">
  <port name="EasterDateSoapPort" binding="tns:EasterDateSoapBinding">
    <soap:address location="http://www.stgregorioschurchdc.org/cgi/websvccal.cgi"/>
  </port>
</service>
</definitions>
```

Notice that all the parts of the message body are described in this document. Also note that, even though this document is intended to be primarily read by a computer, it is still relatively easy for a person with some programming knowledge to follow.

[Try an example SOAP project in SoapUI](#)

WSDL

[The WSDL](#) defines every aspect of the SOAP message. It is even able to define whether any element or attribute is allowed to appear multiple times, if it is required or optional, and can even dictate a specific order the elements must appear in.

It is a common misconception that the WSDL is a requirement for a SOAP service.

SOAP was designed before the WSDL, and therefore the WSDL is optional. Although, it is significantly harder to interface with a web service that does not have a WSDL.

On the other hand, if a developer is asked to interface with an existing SOAP web service, he only needs to be given the WSDL, and there are tools that do service discovery - generate method stubs with appropriate parameters in almost any language from that WSDL.

Many test tools on the market work in the same way - a tester provides a URL to a WSDL, and the tools generate all the calls with sample parameters for all the available methods.

Critique of SOAP

While the WSDL may seem like a great thing at first – it is self documenting and contains almost the complete picture of everything that is required to integrate with a service – it can also become a burden.

Remember, the WSDL is a contract between you (the provider of the service) and every single one of your customers (consumers of the service).

WSDL changes also means client changes.

If you want to make a change to your API, even something as small as adding an optional parameter, the WSDL must change. And WSDL changes also means client changes - all your consumers must recompile their client application against this new WSDL.

This small change greatly increases the burden on the development teams (on both sides of the communication) as well as the test teams. For this reason, the WSDL is viewed as a version lock-in, and most providers are very resistant to updating their API.

Furthermore, while SOAP offers some interesting flexibility, such as the ability to be transmitted over any transport protocol, nobody has really taken advantage of most of these.

There are new advances, but most of these are being hampered by infrastructure routers refusing to route non-standard IP/TCP frames. Just consider, how long has the world been trying to switch over to IPv6?

There is definitely a need for a more lightweight and flexible model [than SOAP].

Any situation where the size of the transmitted message does not matter, or where you control everything end-to-end, SOAP is almost always the better answer.

This applies primarily to direct server to server communication, generally used for internal communication only within the confines of one company.

However, there is a need for a world where almost every person on the planet has several low-memory, low-processing-power devices connected to multiple services at all times, there is definitely a need for a more lightweight and flexible model.

REST

REST – REpresentational State Transfer – is quickly becoming the preferred design model for public APIs (you can learn much more about REST and how to test these APIs in this [REST 101: The Beginners Guide to Using and Testing RESTful APIs Ebook](#)).

REST stands for Representational State Transfer. It is a software architecture style that relies on a stateless communications protocol, most commonly, HTTP. REST structures data in XML, YAML, or any other format that is machine-readable, but usually JSON is most widely used. REST follows the object-oriented programming paradigm of noun-verb. REST is very data-driven, compared to SOAP, which is strongly function-driven. You may see people refer to them as RESTful APIs or RESTful web services. They mean the same thing and can be interchangeable.

There is no standard for the description format of REST services (you can import your REST service in SoapUI by using WADL files). ReadyAPI supports the OpenAPI, Swagger and RAML formats.

Your basic REST HTTP requests are: POST, GET, PUT, and DELETE. SoapUI supports HEAD, OPTIONS, TRACE and PATCH requests as well. Let's look at an example from the [Swagger Pet Store API](#):

- Sending a GET request to /pet/{petId} would retrieve pets with a specified ID from the database.
- Sending a POST request to /pet/{petId}/uploadImage would add a new image of the pet.
- Sending a PUT request to /pet/{petId} would update the attributes of an existing pet, identified by a specified id.
- Sending a DELETE request to /pet/{petId} would delete a specified pet.

So in a nutshell here is what each of these request types map to:

GET	Read or retrieve data
POST	Add new data
PUT	Update data that already exists
DELETE	Remove data

To learn more about REST requests and how to do them in SoapUI, please visit our [Working with REST Requests](#) page.

Example:

A sample message exchange could contain as little as this -

Request:

```
GET http://www.catechizeme.com/catechisms/catechism_for_young_children/daily_question.js HTTP/1.1
Accept-Encoding: gzip,deflate
Host: www.catechizeme.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

Response:

```
Server: Apache
X-Powered-By: Phusion Passenger (mod_rails/mod_rack) 3.0.17
ETag: "b8a7ef8b4b282a70d1b64ea5e79072df"
X-Runtime: 13
Cache-Control: private, max-age=0, must-revalidate
Content-Length: 209
Status: 200
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: js; charset=utf-8
{
  "link": "catechisms\\catechism_for_young_children\\questions\\36",
  "catechism": "Catechism for Young Children",
  "a": "Original sin.",
  "position": 36,
  "q": " What is that sinful nature which we inherit from Adam called?"
}
```

As is already expected this message was sent over HTTP, and used the GET verb.

Further note that the URI, which also had to be included in the SOAP request, but there it had no meaning, here actually takes on a meaning. The body of the message is significantly smaller, in this example there actually isn't one.

A REST service also has a schema in what is called a WADL – Web Application Description Language. The WADL for the above call would look like this:

```
<?xml version="1.0"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xml:lang="en" title="http://www.catechizeme.com"/>
<resources base="http://www.catechizeme.com">
<resource path="catechisms/{CATECHISM_NAME}/daily_question.js" id="Daily_question.js">
<doc xml:lang="en" title="Daily_question.js"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="CATECHISM_NAME" style="template" type="string"/>
<method name="GET" id="Daily_question.js">
  <doc xml:lang="en" title="Daily_question.js"/>
  <request/>
  <response status="200">
    <representation mediaType="json" element="data"/>
    <representation mediaType="js; charset=utf-8" element="data"/>
  </response>
</method>
</resource>
</resources>
</application>
```

The WADL uses XML syntax to describe the metadata and the available actions. It can also be written to be as strict as the WSDL: defining types, optional parameters, etc.

[Try an example REST project in SoapUI](#)

WADL

The WADL does not have any mechanism to represent the data itself, which is what must be sent on the URI. This means that the WADL is able to document only about half of the information you need in order to interface with the service.

Take for example the parameter CATECHISM_NAME in the above sample. The WADL only tells you where in the URI the parameter belongs, and that it should be a string.

However, if you had to glean the valid values for yourself, it would probably take you quite a long time. Note that it is possible to add a schema to the WADL, so that you can define even complex variable types such as enumerations; however, this is even more rare than providing a WADL.

The WADL is completely optional.

Further the WADL is completely optional; in fact, it is quite rare that the WADL is supplied at all! Due to the nature of the service, in order to make any meaningful use of it, you will almost undoubtedly need additional documentation.

Critique of REST

Having a very small footprint and making use of the widely adopted HTTP standard makes REST a very attractive option for public APIs.

Coupled together with JSON, which makes something like adding an optional parameter very simple, makes it very flexible and allows for frequent releases without impacting your consumers.

Arguably, the biggest drawback is the WADL – optional and lacking some necessary information. To address this deficiency, there are several frameworks available on the market that help document and produce RESTful APIs, such as [Swagger](#), [RAML](#), or [JSON-home](#). Swagger has been donated to the Open API Initiative and is now called OpenAPI (OAS). Head over to [Swagger.io](#) where you can read more about this [standard, the specification, and how the Swagger](#)



NO ONE KNOWS API IS BETTER than SMARTBEAR. Find out what our [10 version of SoapUI](#) can do to improve your testing.

Read Next:

[SOAP vs REST Infographic](#)

[API Testing 101](#)

[The Gap Between Goals & Reality in Testing](#)

Open Source

[About SoapUI](#)
[Download](#)
[Tutorials](#)
[Features](#)

Docs

[REST Testing](#)
[SOAP Testing](#)
[Functional API Testing](#)
[API Load Testing](#)
[Security Testing](#)
[Mocking](#)










Pro Tools









[ReadyAPI](#)
[Store](#) ↗

More

[Community](#) ↗
[ReadyAPI Support](#) ↗
[Training & Certification](#)
[Contact Us](#) ↗

Explore SmartBear Tools

 [AlertSite](#) ↗
 [AQTime Pro](#) ↗
 [BitBar](#) ↗
 [Capture for Jira](#) ↗
 [CrossBrowserTesting](#) ↗
 [ReadyAPI](#) ↗
 [SoapUI](#)
 [Swagger](#) ↗
 [SwaggerHub](#) ↗

 [Collaborator](#) ↗
 [Cucumber for Jira](#) ↗
 [CucumberStudio](#) ↗
 [LoadNinja](#) ↗
 [TestComplete](#) ↗
 [TestEngine](#) ↗
 [TestLeft](#) ↗
 [Zephyr](#) ↗

[About Us](#) | [Careers](#) | [Solutions](#) | [Partners](#)

[Contact Us](#) ✉ | +1 617-684-2600 USA | +353 91 398300 EUR | +61 391929960 AUS

