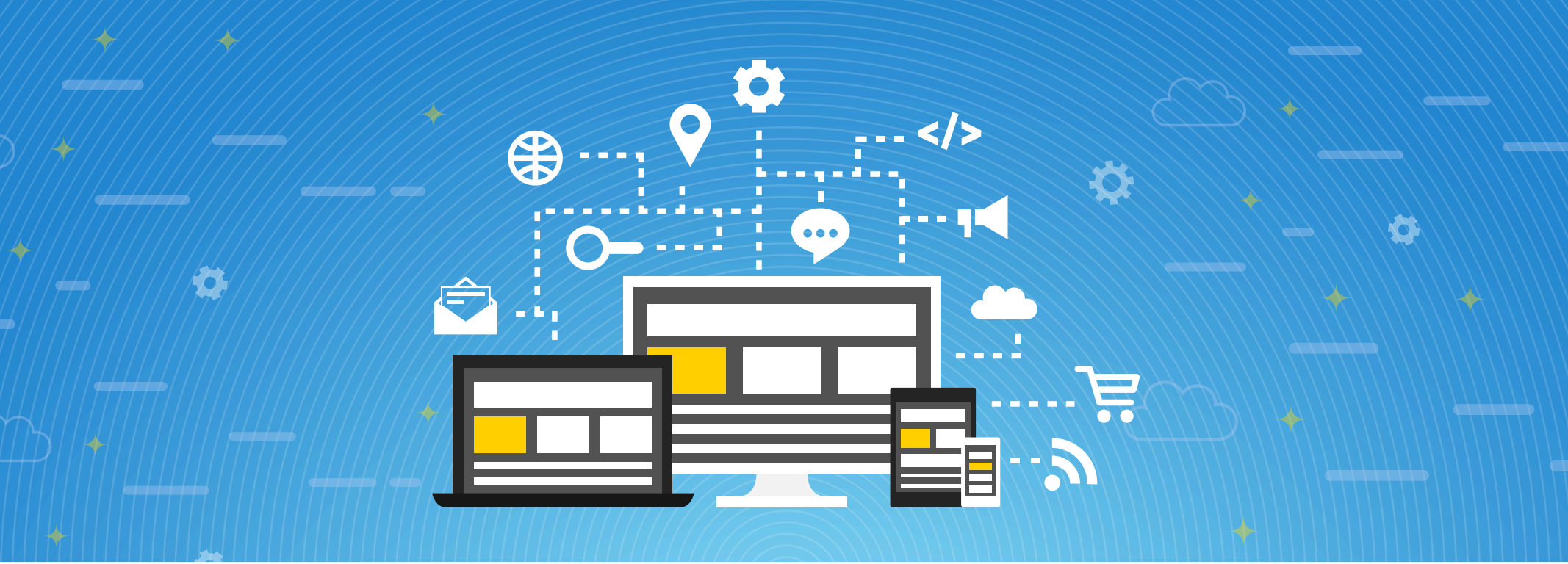


REST 101: The Beginner's Guide to Using and Testing RESTful APIs

SoapUI Pro

SoapUI Pro

SoapUI Pro is the “Swiss Army knife” of API testing that lets developers and testers rapidly create and run automated functional. regression, and data-driven tests for SOAP and REST APIs in one environment.

LEARN MORE ABOUT **SOAPUI PRO**

Contents

You could call this your very own **REST wiki**, except that instead of crowdsourcing the results, we chose to talk to experts on **REST architecture.**

In the SmartBear **REST API tutorial** you will learn what exactly are **RESTful Web Services** and what are its best (and worst) use cases, the difference between a REST API and a SOAP API, and how to test a **REST API** for not only usage, but use cases.

Since this is your very own REST wiki, we will be organizing it around what REST API influencers believe are the most important things for you, as a developer, to know in order for you to build beautiful **RESTful Web APIs** that will actually solve your API consumers' needs.

We'll also show how you can test a REST API with SoapUI Pro, the world's most trusted API testing tool. For REST, SOAP and other popular API and IoT protocols, SoapUI NG Pro provides the industry's most comprehensive and easy-to-learn functional testing capabilities. Along with this eBook, you can download a free trial of SoapUI NG Pro and the other API testing tools as part of SmartBear's Ready! API platform.

Let's get started!

What is a REST API?

Imagine if every PhD dissertation resulted in something that changed the world? Sadly, most end up with a copy on the shelf at the university library, maybe one in the author's office, and little more. But one, about 16 years ago, led to the foundation of that thing we spend our lives on — the Web. Back in 2000, Roy Fielding presented his doctoral dissertation at University of California-Irvine on the representational state transfer.

Representational state transfer or "REST" is the software architectural style designed for distributed systems and, particularly, the World Wide Web. Throughout this REST API tutorial, you will find the same refrain: REST is not a protocol or standard. REST architecture is simply following certain guidelines for how a well-designed Web app behaves, in a logical organization that involves a series of links — or state transitions — that then result in the next page —representing the next state of the application — for the user.

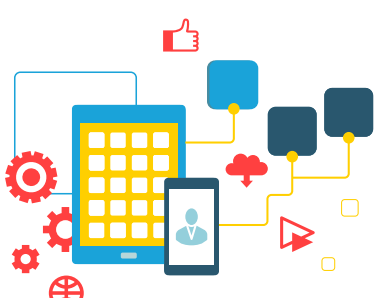
REST architecture is inherently simple because it is based on seven descriptive properties:

- **Performance -** how components interact affects performance
- **Scalability -** able to support large numbers of components
- **Simplicity -** between interacting interfaces
- **Modifiability -** of components to meet changing needs
- **Visibility -** clear communication between components
- **Portability -** of the data-filled code
- **Reliability -** or resistance to fail at system level

A RESTful Web service also has to meet the following architectural constraints, which in turn allow it to have any or all of the desired properties mentioned above.

The **uniform resource constraint** is most inherent to being RESTful design. This constraint decouples and **simplifies** the REST architecture allowing it to **scale** and **modify** independently, increasing portability, **visibility**, and **reliability** of the components. The four constraints found within the uniform resource constraint are:

1. **Identification of resources as 'requests'**, in a simple way that is understood independent of original language or interpretation. (We'll "get" more on this in a bit.)
2. **Manipulation of resources.** When a client has a representation of data, it can then modify or delete that resource.
3. **Self-descriptive messages.** Self-descriptive in itself; this kind of messaging is complete as an entity and in itself can describe how it can be processed.
4. **Hypermedia** means that no other actions will be assumed besides those self-described.



If it doesn't meet the uniform interface constraints nor the following constraints, it can't be deemed a RESTful system:

- **Client-server separation** - These two are completely separate, enabling them to be developed and replaced independent of each other. Client code becomes more **portable** because it is separated from data storage, while the server becomes more **scalable** because it is unconcerned with user state or interface.
- **Stateless** - Each data request and response pair is treated as completely independent from prior and future requests. These states are "in transition" when one or more requests are outstanding.
- **Cacheable** - Everything on the Web can be cached, so responses must clearly define whether or not they are cacheable, avoiding either inappropriate caching or caching of old information.
- **Layered systems** - Intermediary servers must be available to make the system more **scalable**.
- **Code on demand** - This is the only "optional" REST constraint. Servers can temporarily extend or customize the functionality of a client by the transfer of executable code, like JavaScript.

Everything in the RESTful architecture is about resources. A resource is an object with its own associated data. Resources have relationships with other resources and a set of methods or verbs to operate between these resources. Then you can have a collection of resources which can interact as a collection with one or more resources or collections.

REST is simple as a concept because it follows a basic language of HTTP 1.1 hypertext transfer that the entire Web understands, namely the following, self-explanatory action verbs, which are usually written in capital letters to stand out:

- **POST** - to add data, like to a message board
- **GET** - to retrieve data, but without altering it, from a particular URL
- **PUT** - to save an update to the unified resource identifiers (URI)
- **DELETE** - to remove a specified resource
- **PATCH** - to make a change in a request

Now, with only these limited operations, REST **simply** focuses on interactions between data elements and on what roles components play, rather than focusing on details like language and implementations.

REST became the basis on which HTTP standards and URIs were designed, which were also developed by Fielding in parallel. Bringing all three things altogether and REST easily became the prevailing and accepted software architectural style for the World Wide Web — not too shabby for a PhD dissertation!

EDITORIAL NOTE: Putting grammar to REST

We will use terms like **REST definition** and **RESTful definition** somewhat interchangeably throughout this eBook. There are nuanced differences between them but truly, at this point, it's all grammar: REST is the more common noun and RESTful — not the commonly misspelled "RESTfull" — is an adjective used to describe code conforming to the qualities of REST. Particularly in the world of the application programming interface, the terms REST API and RESTful API mean pretty much the same thing — a RESTful API adheres to the RESTful architecture constraints.

RESTful Web services vs. SOAP architecture

"This is a RESTroom, no SOAP allowed."

API offices seem to always sport this pun on the door of their facilities. It's not that developers are gross, they are just punny. SOAP and REST Web services have had a theoretical war raging for a while now. Just as RESTful Web services are not the be-all and end-all solution for every architectural use case, neither is the Simple Object Access Protocol.

But where do REST and SOAP differ? Some would call this comparing apples and oranges —after all, SOAP is an actual protocol, while REST is just a set of constraints. And Microsoft, for one, has long distanced itself from the competition because it doesn't feel it has to be the battle of SOAP vs. REST. But, since software teams are choosing to make the distinction and the decision between the two, we certainly couldn't address RESTful architecture without addressing SOAP architecture.

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.

SOAP-based APIs and SOAP Web services tend to run more expensive than other contemporary options. SOAP relies on building XML-based systems, which means the amount of data is inherently larger, which in turn means it'll cost you a lot more for central processing unit (CPU) and memory usage, usually to the point that you have to build custom servers to handle the load.

This is probably why, according to at least one source, while SOAP used to dominate the API space, now 70 percent of all public APIs are REST APIs.

At Mashape open-source API management tools, Ahmad Nassri says, “Everything we have ever done is in the REST space — not SOAP or WS Deathstar,” referring to the common joke around Web services being on their way out, waiting to take down the Force of your operations.

There are other reasons that Mashape goes REST all the way. “RESTful design means that it is stateless, cacheable, and you can put a test around it,” said their VP of engineering, of REST’s renowned flexibility.

And probably one of the main reasons Mashape builds RESTful Web services is because they cut the cost of development, deployment, and maintenance. “With REST, not only are you bypassing that cost, but you’re bypassing the need for those custom servers actually being built for SOAP. REST is portable and friendly and cheaper to get started,” Nassri said.

He went on to point out the benefit of RESTful Web services being actually built around the HTTP protocol itself. With SOAP, “you have all this terminology and verbiage that you have to parse into the payload itself, to parse the body of the message to understand what the intention of the message is, and [you have] to do a lot of processing just to figure out what needs to be done.” An HTTP server is built to receive verbs like post, pull and get which means everything RESTful simply works — at the server level, at the HTTP level, and at the level at which the developer can understand.

To put it more succinctly: The REST API simply works because it was built with the Web in mind, because it was built on top of the Web.

While SOAP architecture was also built in the early days of the Web back in 1998, it was designed by infrastructure giant Microsoft and built for XML. The REST architecture was developed between 1996 and 1999 in parallel to the HTTP protocol, making it almost always the default looked to in terms of the next generation of APIs and standards.

RESTful Web services were built on the foundation of the Web to understand things like caching, proxying, and client-server relationships. These are things that every single browser, Internet service provider (ISP) and proxy can understand.

Nassri pointed out: “When you build your API on the same foundation, you’re already leveraging all the infrastructure.”

This means, unlike SOAP, you don’t have to do anything to translate or interpret. For example, to cache, you only have send a message to the server that a particular set of content needs to be cached and then it is cached efficiently. This all makes REST much easier to code in and write API documentation for.

“As a developer, whether you’re a startup or a company or big business you have a product, you want to prototype an API and test it, build it and ship it,” Nassri said. “The last thing you want is to spend months building the tools to build your API. You just want to build your API. And this is what REST APIs are really good at—just leverage the technology and build.”

This doesn’t mean SOAP is by any means dead — many companies, including Salesforce and Paypal, find comfort in the fact that SOAP is a set of protocols and has the infrastructure to back it up. Also, since REST is inherently stateless, SOAP, with its stateful operations, is better suited to support conversational state management.

Plus, the flexibility of REST actually doesn’t preclude you from using SOAP, since you can apply the RESTful concepts to any protocol. In fact, while mostly associated with HTTP, REST representations are increasingly based on JSON, URI or XML standards. Conversely, however, if you follow the full protocols of SOAP, you can’t really go REST.

Of course a lot of companies are going REST over SOAP because it’s simpler and repeatable.

When you decide to write code against a RESTful API, you’ve already gone down that rabbit hole so you already have the core competency for REST, so the next REST API you use, you’ve already gone down that learning curve so you can most likely reuse a lot of that code,” SendGrid email delivery software’s Matt Bernier said. “Basically doing the same thing over and over again, just a different URI, or data structure. All you need to know is what the variations for each specific endpoint you’re calling are.”

This means that REST APIs are set up really well for API testing automation.

A standard by any other name would smell as sweet

To raise a controversial question: Is REST really not a standard? Technically no, which causes controversy unto itself that we'll talk about later, but should we respect it like one?

If you talk to Bernier, we should treat REST as a set of standards. While not wanting to ruffle any feathers he said, "I'm going off the definition of a standard being a commonly agreed upon set of rules to do some sort of task. It's not an RFC [request for comments] specification, but REST is well agreed upon as a set of rules to follow—then it's a standard," he says that just hasn't been made official for an RFC.

Bernier pointed out that a lot of people refer to it as the "REST spec." He went onto comment that "REST is kind of like agile: you take the guidelines and make it your own but you're still communicating with your own words." Plus, there's no doubt that there are RESTful constraints, specifications that you just don't cross.

But maybe it doesn't even matter if people are calling it a standard or not. With the vast majority of companies going toward REST, maybe, standard or not, it bears consideration as a solution for your business. And so we continue the talk of REST APIs after that semantic rest stop.

When is a REST API a Good Idea for You and Your Customer?

Just like everything, REST isn't necessarily the perfect fit for you or your end API consumer.

Nassri says REST is the best when information needs to be read and written at a resource level, like:

- a LinkedIn profile
- photos online
- reading data submitted online
- submitting data online
- reading information from a server or database
- ordering information

"The application needs access to that object so it can know and remember where that object is and how to get to it. And to continue to query that object or get that object throughout the application lifecycle." Nassri clarified this statement with: "I say application not user — when people talk about REST APIs and APIs in general, they talk about how the APIs should be designed for the developer — I somewhat disagree — it needs to be readable by the developer but ultimately it needs to be designed application-friendly. If the application isn't able to leverage the API efficiently, then the API fails."

He says this because the RESTful design by approach is useful for applications, which in turn makes for an API that is more successful for the developers.

"We talk about RESTFul designs and RESTful architecture of that design for the API provider, but we should flip that and talk about the people we are building the APIs for, more than the people building it. If nobody's using it or it's too hard for them to use it, well, the API becomes irrelevant," Nassri said.

"Because RESTful architecture is not hammered down, they have various interpretations of it, but that's fine as long as the focus is on the end user."

For SendGrid, they never went SOAP but had two other iterations of their email API before, first SMTP with HTTP endpoints, and then Version 2 had HTTP calls that would accept JSON and XML.

"When we made our new API we made it follow REST to match what the industry is doing," Bernier said, of Version 3 which is a purely REST API.

He went onto give an example of how SendGrid's REST API reflects their API legacy and looks to suit the needs of current and future API customers. "Our mail-send endpoint is *v3/mail/send*. Many people would have made it a POST on *v3/mail* and called it good, but we chose POST *V3/mail/send*," explaining that this choice adds to the intuitiveness of the name and it fits into the historical context of previous send endpoints at SendGrid.

He went on to give the example of why SendGrid chose to use *schedules/now*, which is important to the success of SendGrid's new Marketing Campaigns API. "When you want to send a marketing campaign now, you hit POST / *schedules/now*. This way you are deliberately choosing to take the extra action to schedule the campaign for 'right now', rather than accidentally sending before you're ready or at the wrong time."

But just because REST makes it easy, doesn't mean that it's a snap of the fingers to start with.

To do your best, you need to understand more than REST

Still, we want to emphasize that REST and the RESTful architecture is not a standard, but rather it's an architectural style. For this, you have to remember that REST doesn't just need knowledge of the basic principles of REST, but developers also need — and quite possibly already have — a basic understanding of the foundations of the Web on which your REST APIs will run:

- HTTP servers, HTTP proxies, and HTTP caching
- How to Web works on HTTP standards
- URLs
- URIs
- JSON and how it's handled on the Web
- What CPU architecture your computer runs on
- Protocol RFC-20-16, the Uniform Resource Agents standards the Web is built on

Of course, since this is an essential part of Web and even human history, it's good practice to understand the origin of these terms and the basic foundation of

each of them, anyway, so this isn't so much another kind of RESTful constraint but a great learning opportunity!

Of course, like all things with humans involved, there are other challenges.

Are *not* having standards their own constraints?

Besides the basic constraints and properties we described, the rest of REST can be left up to interpretation and expansion, which could be good or bad, depending on the person doing the interpreting.

"The REST space is interesting because it has all kinds of things around the Fielding dissertation. And then you have this entire industry around RESTful technology — not just APIs — pivoting around that paper he published. It's interesting because it's so un-opinionated beyond the foundational elements that the paper describes, leaving the idea of REST wide open for interpretation and expansion," which Nassri says can be good or bad.

He argues that with SOAP and Oasis WS standards, "back in the day, they tried to solve the problem by introducing too many standards. Now we have with REST, too little."

"The state where REST is today is open for interpretation — it's not a standard or a spec, it's a 15-year-old research paper. As smart and comprehensive as it was and still continues to be, it is not as representative of our technology as it is today," Nassri pointed out. "As it's not meant to be a standards paper, and people just interpret things the way they want and have various expectations and interpretations around the software."

When the REST API is *not* the answer

Nassri says that he is often approached to advise developers as to what kind of languages, frameworks and standards they should use. He always turns it around to one question: What problem are you trying to solve?

"I could answer it but that doesn't help people learn nor understand the vastness of the space we're in. You gotta do research about your problem. You gotta understand what your business is. Is your business building APIs? Or is your API a different product than your main product?"

From this understanding, you can then build your use cases—which are almost never what you assume. Nassri gave the example of the media industry, where certainly the entire sports statistics world is still being shared via FTP, even to big names in the API space like Google and Yahoo.

The fact is: REST is not necessarily the answer to everything.

Indeed, it's a bad idea for many things. REST works with the Web because it was written around HTTP standards and the Web back in 2000 well before mobile was even really a concept. Mobile-first apps work hard to avoid APIs because it's expensive and drains the battery when it makes calls to the network.

REST APIs and APIs in general would also never work for the big games like Call of Duty or World of Warcraft, where there's a real-time voice chat system among the players based anywhere in the world. These games avoid APIs like the plague, instead going for really fast and compressed binary data dumps that allow for real-time data transfer, not waiting for API calls.

Testing a REST API: Getting Started

A testing REST API example: Build your own application to give it a try

If you decide a REST API is well-suited to you and your customers' needs, then it's time to design and test it based on those needs.

In order to fully understand how your target audience would use your API, you have to dogfood it yourself and then, once it's out in the wild, you need to monitor how it's being used. For both API testing requirements, let's use the common **REST API example** of a simple contact system or address book.

You can start by assuming that there is some sort of contact information as endpoints—a name, an email address, a phone number. Then, you build an application around that. If you design your endpoint so you make an API call for contacts first, then you have to make subsequent calls to the email and address endpoints. For an address book of ten addresses, it is a minimum of 30 REST calls just to have a visual representation—like an address book—of your application.

Building an app just to test out your API may seem expensive, but you don't need to produce a commercial-ready app, but rather one that just has the parts that the consumer wants to use to interact with your API. "Doing things like this helps you understand the architectural design of your API," Nassri said.

And like all products, "When it comes to testing, you need to figure out your use cases of what people are building around your APIs," not just the standards or what data goes in or out. A lot of people look at APIs as data in and data out, but they should be a lot more than that."

MUST READ:
SOAP API Testing for Beginners

REST API testing for developer experience

Hitch developer community portal co-founder Bruno Pedro told SmartBear that one value of the REST API is in creating and automating functional testing. “Testing REST APIs lets you confirm that a controlled input always generates predictable responses.” He went on to say that in order to have successful functional testing, there are certain requirements:

1. Know which API calls to test and how your application uses them.
2. Identify the input you need to provide to make tests meaningful.
3. Using a tool like Ready! API, generate input using fake data that resembles what a real user would do.

But to be able to do this right you need to understand how your API consumers are going to use your API.

“When you’re using any product or tool to test your APIs, the tests you’re building are not just to validate the input or output with the API, but what applications can be built on it,” something that Nassri says is often missing.

When asked how do you do that when you are first building your API and maybe don’t know who your users are, he recommended to just create a hypothesis of the end user experience and adjust from there. He says that when you’re building API tests, it’s not enough to know the endpoint, “You actually have to build an application around it to test it from a usability point of view.”

Nassri offers the example of Flickr. This image hosting site was built as a photo service with a target photographer audience, but the Flickr API has nothing to do with that audience.

“Look at your API design as a product, how does that work and how does that affect your audience? The user experience and how users use it.”

Another important thing for REST API testing — beyond response time and accuracy of data — is making sure you also have API analytics and reporting. Testing is not just data in, data out, it’s testing how your API can be used and then after you’ve launched it and begin to use it, you can further change and update your API in response to users’ needs.

You also need to understand how your end-user developers are using your API. Nassri says he wants to understand:

- When developers are using the API
- Which endpoints they are using
- Which endpoints they are using more than the other(s)
- When they use one particular endpoint
- Why they use this particular endpoint
- What combination of endpoints they are calling

Continuing with the same REST API example as offered before, by knowing that when people are making REST calls for */contacts*, they always also call the */email* and */address* endpoints, you can optimize your REST API so that these three endpoints are consolidated into one API call. So whether you are going the way of building your own application to test out your RESTful API use cases or you are carefully monitoring the API usage with an API testing tool, you uncover a need to make changes to your REST API which reduces the amount of API calls that need to be made.

Nassri points out that this is win-win — you make your API consumers happy because they have to make fewer calls, and your DevOps rejoice because it's less strain on your servers.

The Three Levels of API Testing

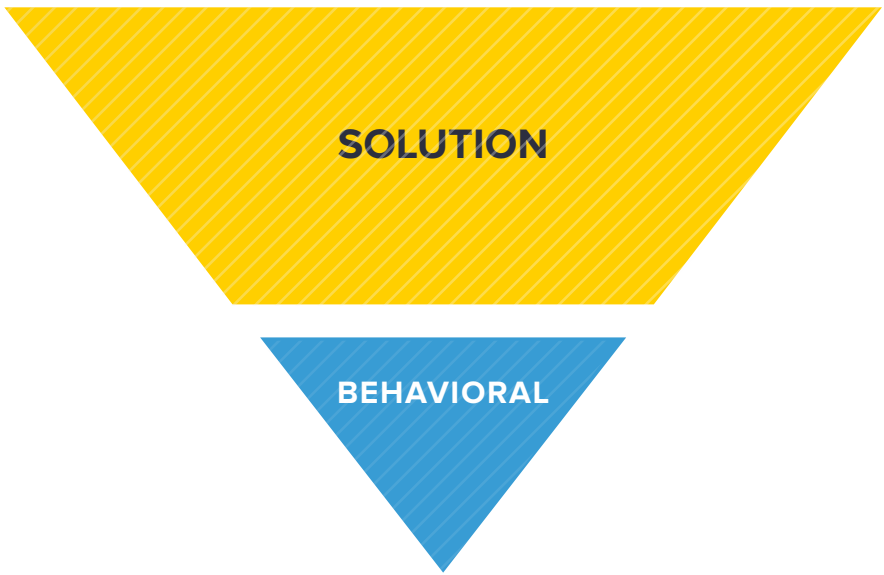
"APIs, by their nature as being over-the-wire [or network protocol], allow for testing at a variety of levels: behavioral, contractual, and solution-oriented." API architect and founder of LaunchAny API strategy and design agency James Higginbotham breaks down API testing into these three essential aspects:

- **Behavioral API testing** ensures that it delivers expected behavior and handles unexpected behavior properly. This is the lowest, most internal value. Behavioral testing ensures that the REST API delivers on the expected behavior and handles unexpected behavior properly.
  Does the code work?

- **Contractual API testing** ensures that what is specified by the definition is what has actually been shipped via code. This falls at the middle level of needs. Contractual testing ensures that what is specified by the definition is what has actually been shipped via code.
  Does the API contract continue to function as we have defined it? With the right inputs? Outputs? Data formats?
- **Solution-oriented API testing** ensures that the API as a whole supports the intended use cases that it was designed to solve. This falls as the highest, mostly external value. Solution-oriented testing ensures that the API as a whole supports the intended use cases that it was designed to solve.
  Does the API solve real problems that our customers have? Does it do something that people actually care about?

Higginbotham calls this **Maslow's Pyramid of API needs**, made up of growing concerns from the lowest—the internal—to the highest—external—levels. "Teams need to look beyond just testing for functional and behavioral completeness. They need to move upward to ensure what they are externalizing to internal and/or external developers is complete."

Maslow's Pyramid of API needs



Higginbotham says to flip that pyramid upside-down, balanced with the behavioral API testing tip as the base for all API testing. "I'm suggesting that if you flip the pyramid with API testing, you can focus more on testing the solution as it can be automated, unlike browser and mobile apps. Thus, you cause your QA team to become more focused on verifying value to the customer, with the other testing at the other two levels focused on internal workings to catch bugs in isolation."

He points out that the solution-oriented API testing is often the hardest to carry out, particularly when it is a mobile or browser-based app. He says that when the data is run through a REST API, this testing becomes much easier and can add tremendous value.

“The constraints of REST encourage the use of HTTP specification, making it easier to build out tests by composing the right requests and responses, without the more complicated SOAP-based protocol details involved,” Higginbotham went on to say.

“The testing world has always pushed for the lower-level tests that verify internal functionality to the detriment of focusing on the product deliverables, because much of the UI-focused testing requires manual effort.” But, he says, this manual user interface testing is essential for putting developer experience first.

What you need to know for testing REST APIs

The basics of good API testing — and good software testing really — are the same — you have to do it early, often and continually, and much of it can be automated.

What makes testing REST APIs particularly easy is because of the clear relationship between the data, made even clearer when used in conjunction with Hypermedia APIs.

“Again in Hypermedia there are a lot of standards that you can follow, but that also helps the hypermedia descriptions between the entities of the API or the data,” Nassri explained. “If you are following a RESTful architecture, you are sending descriptions from areas. With hypermedia, you’re describing the relationships between developers. Hypermedia can become a very important part in terms of testing APIs.”

He says Hypermedia is used a lot in terms of testing the functionality of APIs, but he says it is not used enough in terms of vesting and validating the behavior of the API, like they do at Mashape, with everything based on the HTTP spec and URLs, taking part in their APIs being very RESTful by design.

Bernier echoes Nassri by saying that since REST is very standards compliant, it means you have description languages that can confirm to that as well, like Swagger, which SendGrid uses. He says REST is a small subset of specifications for how to send and accept API calls. When combined with other standards, this is where he finds an ease for API testing.

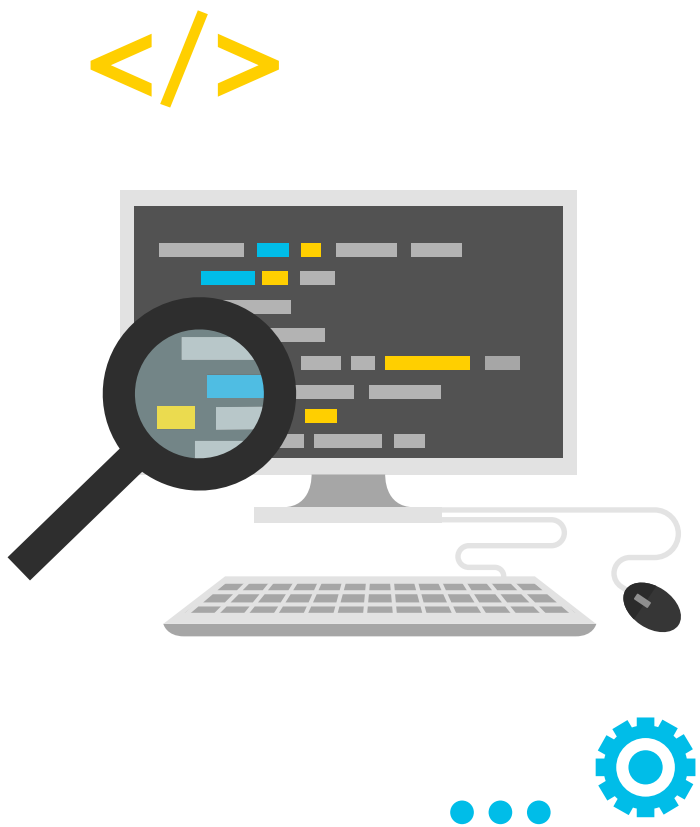
“You take a description language, like Swagger, that is based on the REST specification, is designed to describe REST endpoints, and is written in JSON. [Together they] allow you to then code against the specification and automate against that specification,” he said.

Also because the specification has all of the data about the fields, the structure, and the endpoints and the methods, URLs, that makes it really easy to then write both basic and more advanced tests that can be automated.

Bernier went onto explain: “You have this description language that gives you all the information, and all the variables have been taken out so there are no edge cases. In software if you can take all the edge cases you, it’s pretty easy to automate. And then you can write code against each edge case and then if that code happens to create tests then you have all of your unit tests and integrations tests for your API.”

From this you can also then automate things like documentation and library generation.

“It makes it all really easy,” Bernier said.



REST API Testing with SoapUI Pro

When it comes to testing REST APIs, having the right tools can make all the difference. SoapUI Pro is built specifically for testers and developers to improve service quality and speed time to delivery. The tool includes data-driven input and validation from spreadsheets and databases to ensure your tests are comprehensive, and even run security and coverage reports to make sure the critical aspects of API quality are part of your delivery process.

Other key features of SoapUI Pro, include:

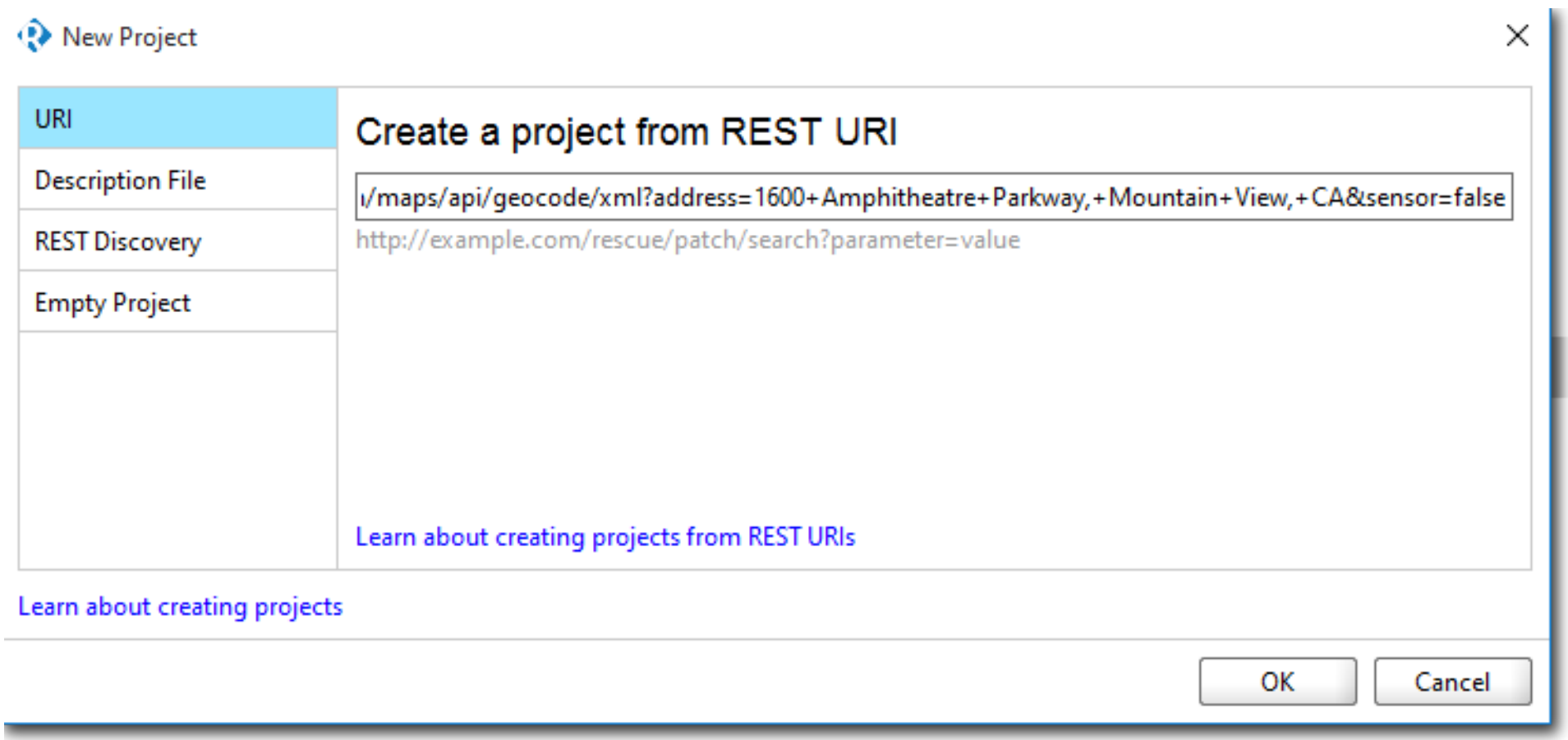
- Create tests directly from Swagger and other popular API description formats
- Analyze your functional test coverage to know what you're missing
- Run ad-hoc tests without having to maintain temporary API client code
- Use the command-line to integrate your tests into your build system
- Quickly flip between multiple environments: development, testing, staging, etc.
- Test REST, SOAP, and other protocols in a single, unified framework

Setting up your first test in SoapUI Pro couldn't be easier. Here's a closer look at how it works:

Getting Started With REST Testing

Start by creating a new REST project from the File menu by choosing the "New REST Project" option in the File menu: Specify the following Google Map API URL in the Service Endpoint Field:
http://maps.googleapis.com/maps/api/geocode/xml?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false

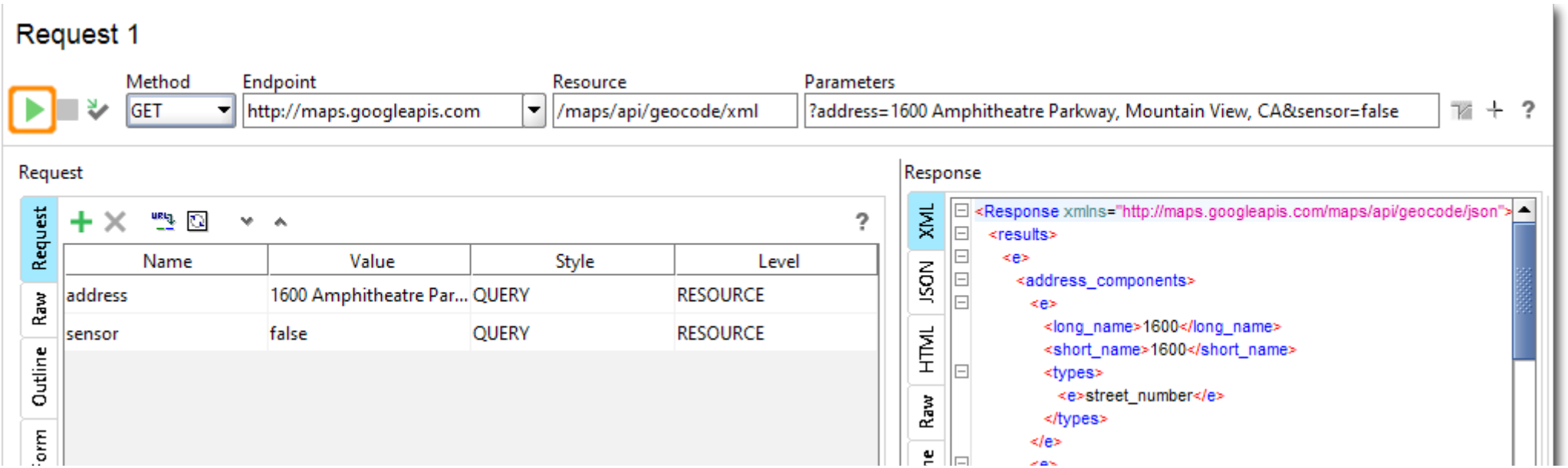
Here you can just Click **OK**, and SoapUI creates the project complete with a Service, Resource, Method and the actual Request and opens the Request editor.



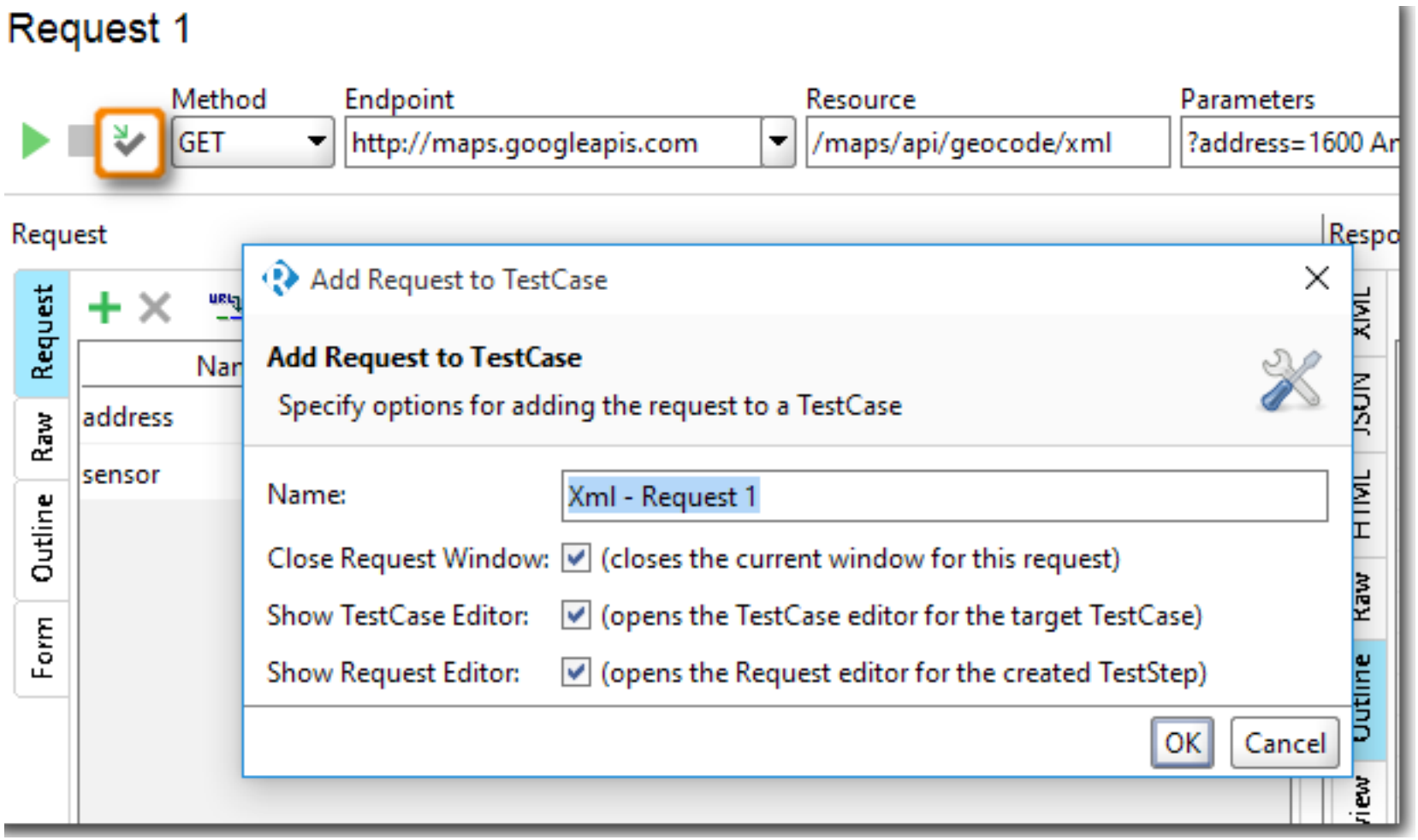
In the “Parameters” table, you can see that SoapUI has automatically extracted the different query-arguments from the path.

Click the green arrow at the top left in the Request editor and you can see the XML output returned by the service:

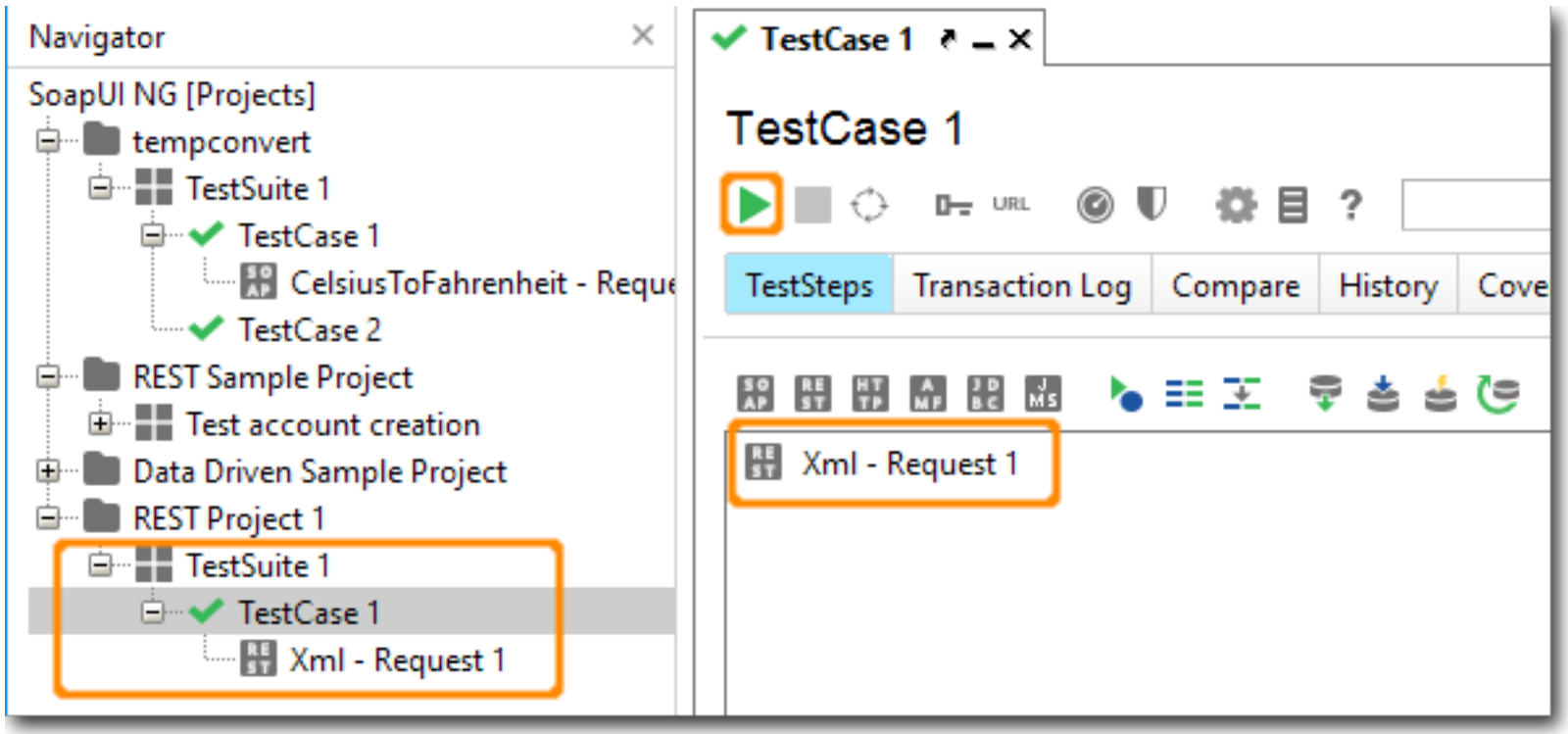
Here you can just Click **OK**, which finally creates the actual request and opens its editor. Click the green arrow at the top left in the Request editor and you can see the XML output returned by the service:



The request seems to be working fine, so we're all set to create an actual functional test for this resource. Click the "Add to TestCase" button at the top left, which prompts for the names of an initial TestSuite and TestCase, then it shows the following dialog:

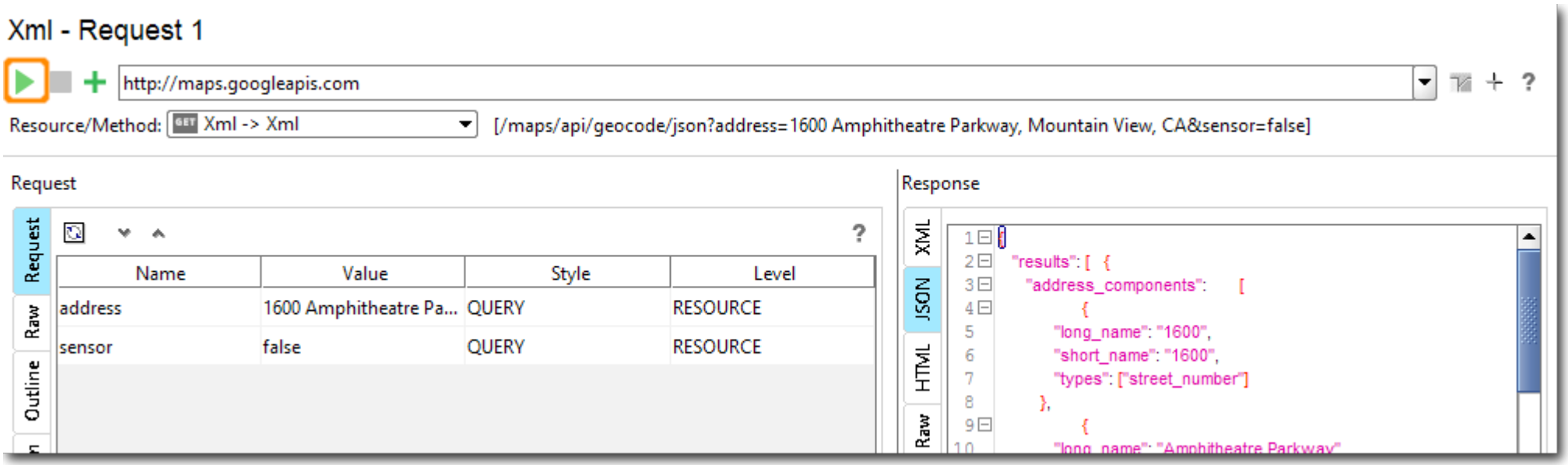


Just go with the default options for now and Click **OK**; SoapUI generates a corresponding REST Request TestStep into the TestCase:



Now double-click the resource icon in the Navigator and change the _Resource Path _to "/maps/api/geocode/json":

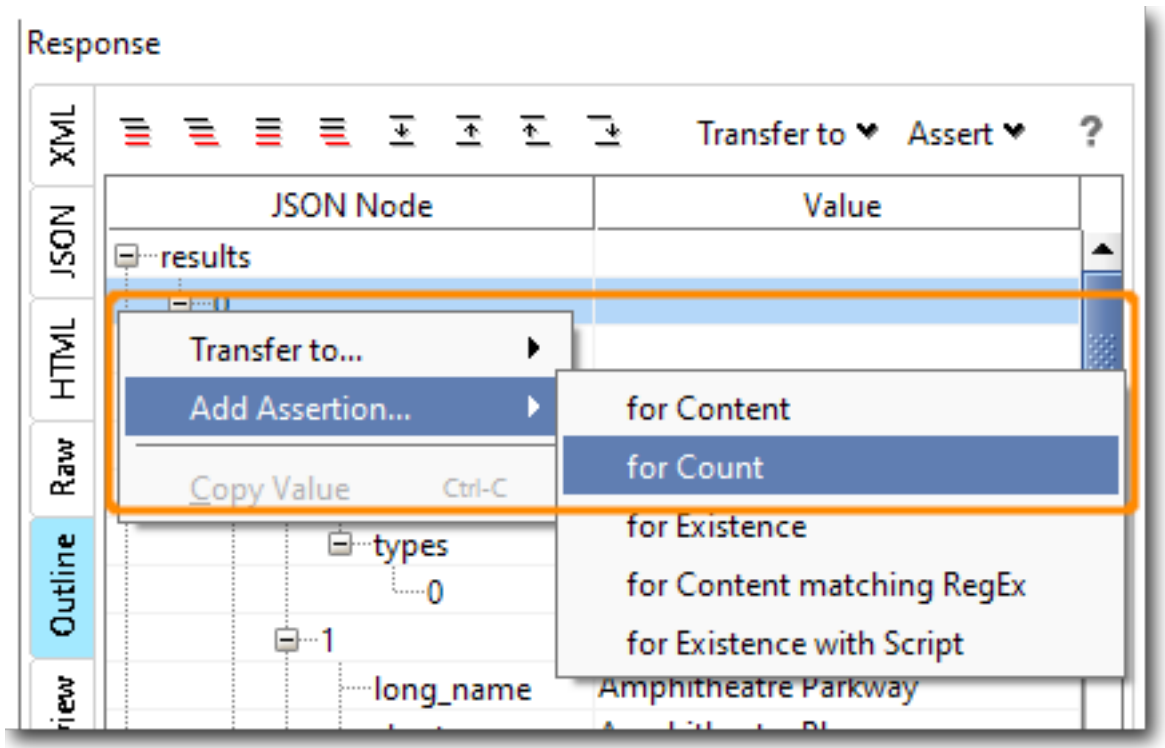
Now go back to the previous request and run it again:



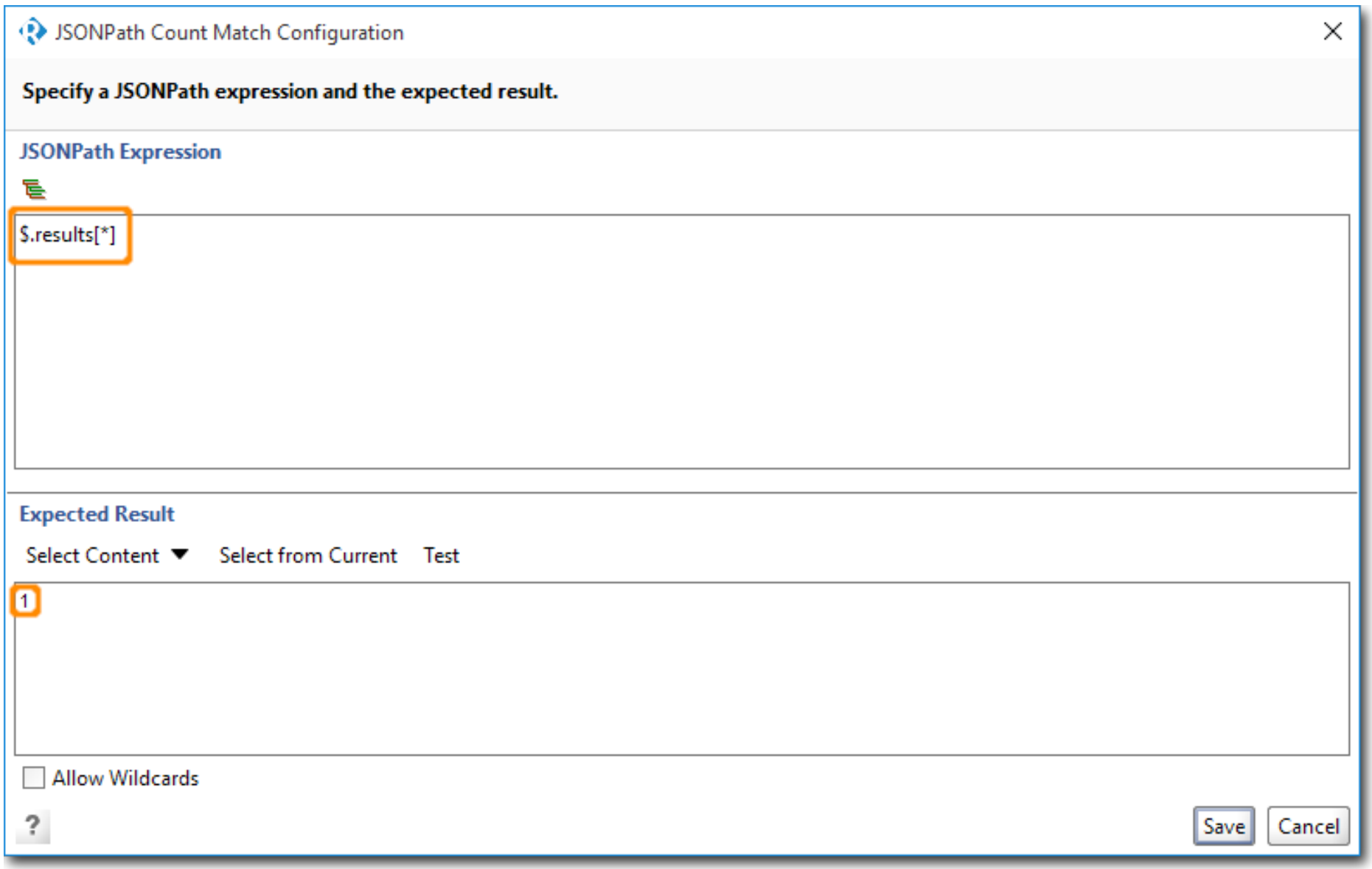
Now you can see a nicely formatted JSON response in the JSON view to the right instead of the previous XML result.

Ok! Time to add an actual assertion to validate the content of the response. In our case we are just going to check that we get 1 place back from the service, open the “Get places - Request 1” TestStep and submit it as usual giving the same JSON response as above. Then in the right response part of the window now select the “Outline” view and right-click on the first “e” item.

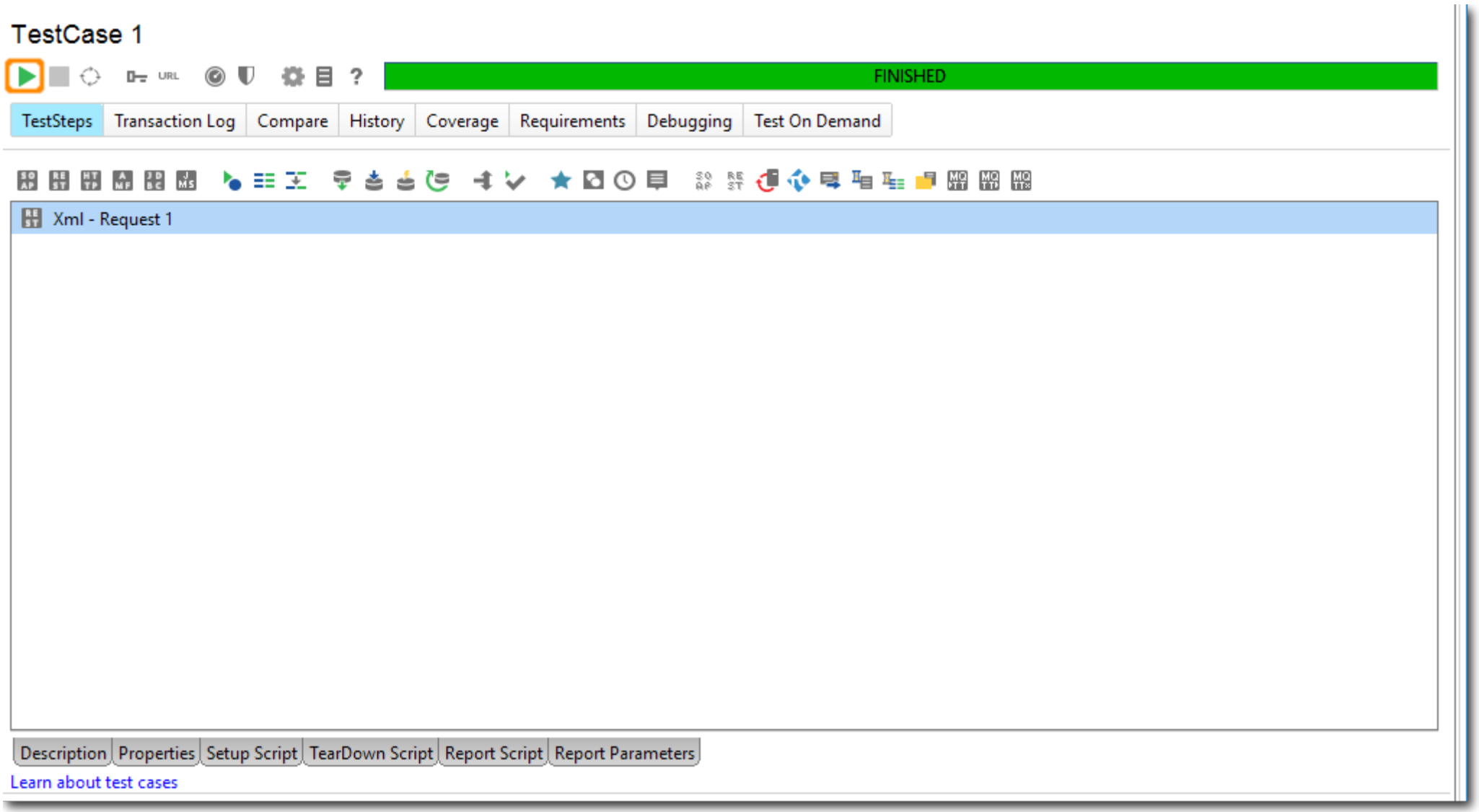
Then in the popup menu, select the “Add Assertion ➔ for Count” option, which automatically generates an XPath assertion for you (this is a SoapUI Pro feature, in SoapUI open source you should create this assertion by hand):



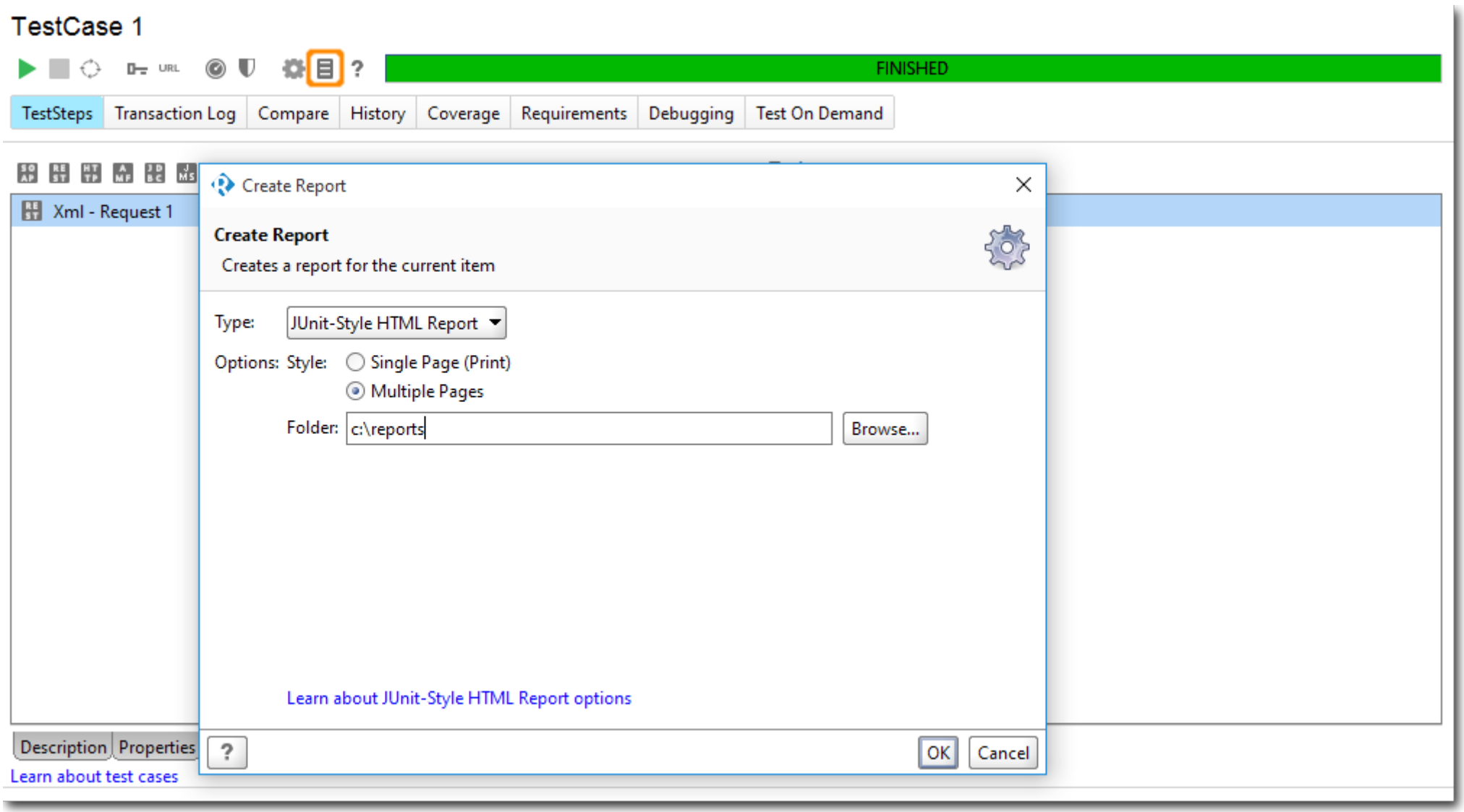
Here you see the generated XPath statement at the top and its expected result below. All is fine, just Save the assertion and go back to the TestCase window:



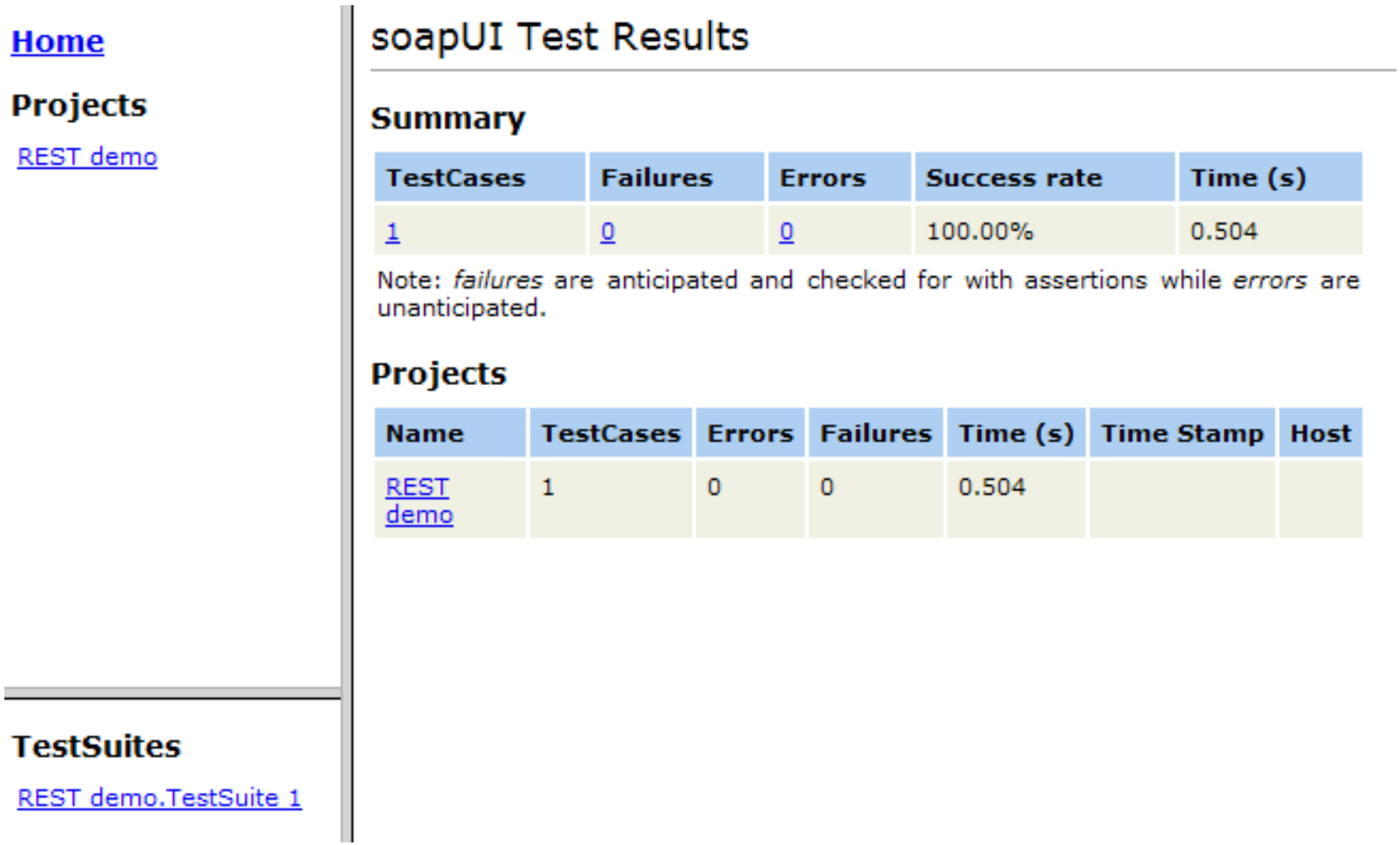
Run the TestCase with the green arrow at the top left which will result in the above output in the Log at the bottom; your functional test passed just fine!



Finally, if you are running Ready! API, you can create a simple HTML report. Click the “Create Report” button in the menu at the top and select “JUnit-Syle HTML Report” in the opened dialog as follows:



Click **OK** and SoapUI will generate the report for and open it in the system browser:



Home

Projects

REST demo

TestSuites

REST demo.TestSuite 1

soapUI Test Results

Summary

TestCases	Failures	Errors	Success rate	Time (s)
1	0	0	100.00%	0.504

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Projects

Name	TestCases	Errors	Failures	Time (s)	Time Stamp	Host
REST demo	1	0	0	0.504		

Voila! Your first functional test of a REST service with SoapUI Pro is just a couple of clicks away.

SMARTBEAR

Over 6 million software professionals and 22,000 organizations across 194 countries use SmartBear tool

6M+ users

22K+ organizations

194 countries

See Some Succesful Customers >>

API READINESS

Functional testing through performance monitoring

SEE API READINESS PRODUCTS

TESTING

Functional testing, performance testing and test management

SEE TESTING PRODUCTS

PERFORMANCE MONITORING

Synthetic monitoring for API, web, mobile, SaaS, and Infrastructure

SEE MONITORING PRODUCTS

CODE COLLABORATION

Peer code and documentation review

SEE COLLABORATION PRODUCTS


SoapUI Pro