

Tutorials[Web Service Sample Projects](#)**[SOAP Sample Project](#)**[REST Sample Project](#)[Data Driven Testing](#)[Twitter](#)[Facebook](#)[Flickr](#)[Home](#) / [Resources](#) / [Tutorials](#) / SOAP Sample Project

Sample SOAP Project in SoapUI

Why Use ReadyAPI for Web Service Testing?

While SoapUI Open Source can be seen as the Swiss Army knife for testing, ReadyAPI is the tool with the sharpest edge. Applied to testing web services, ReadyAPI focuses on enhancing efficiency and usability. With [Point and Click Testing](#), you can drag and drop instead of manually writing the code. [The Form Editor](#) creates a form from your request, eliminating the need for you to spend time on repetitive coding. [The Outline Editor](#) simplifies and exposes the XML structure. These functions make your testing less time-consuming. If creativity, flow and speed are important to you as a tester, ReadyAPI is for you.

[Try ReadyAPI - the world's most powerful web services testing tool](#)

Web Service Sample Project

Getting started with a new application is always a challenge, no matter how complex it is. In order to get started, SoapUI comes with a sample file that can be used as reference and inspiration.

Note: If you chose not to install tutorials when installing the product, you do not have the sample file. Use the SoapUI installer to update the existing installation and get access to the sample file.


The sample illustrates some of the basic concepts of SoapUI and can be used as a starting point. Try out the project, run the included mock service and tests to familiarize yourself with the SoapUI interface, and then move on to the [Your First ReadyAPIject](#) tutorial.

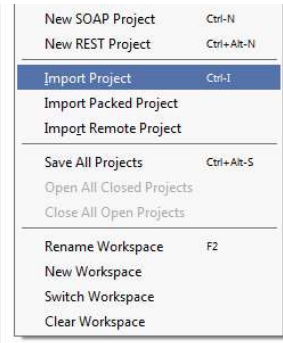
Here are the main concepts illustrated in the ReadyAPIject.

- [Web Service Inspection](#)
- [Functional Tests of Web Services](#)
- [Web Service Load Tests](#)
- [Web Service Mocking](#)

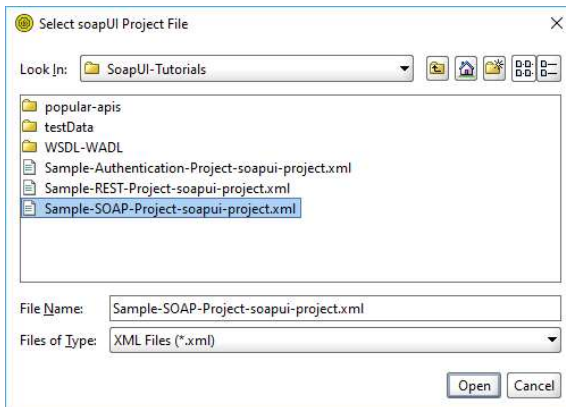
Importing the Project

Let's start by opening the project.

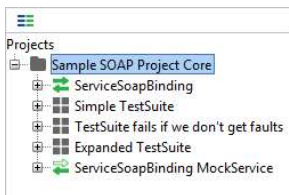
1. Click  on the main toolbar or right-click the root node in the Navigator panel and select Import Project:



2. In the **Select ReadyAPIject File** dialog, select the *Sample-SOAP-Project-soapui-project.xml* file from the <Home directory>/SoapUI-Tutorials folder.



3. The sample project will be shown in the SoapUI Navigator.



The structure of a ReadyAPIject is like this:

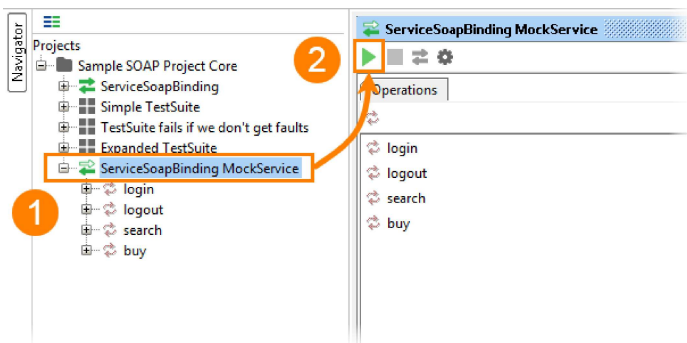
- Project
 - Interface
 - Test suites
 - Mock services

Run mock service

Endpoints of sample requests refer to a mock service. Run the mock service before sending a request to it. Otherwise, the test will fail. To learn more about mock services, see [Web Service Mocking](#) below.

To run the sample mock service, do the following:

1. Double-click the ServiceSoapBinding mock service.
2. In the mock editor, click ►.

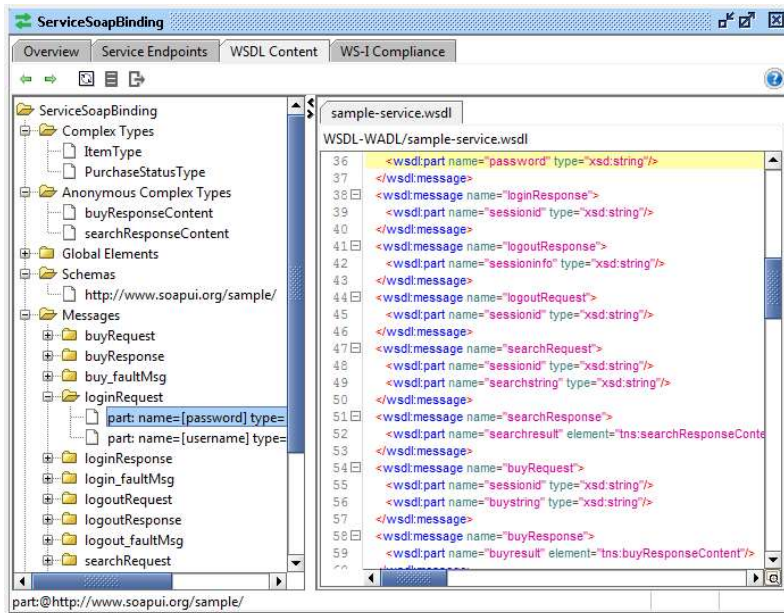


Introduction

Web service inspection can be very helpful at an early stage of the testing process when you want to find out how a web service works. You can do this in two ways: by inspecting the web service's WSDL file and by making web service requests.

Tutorial

1. Double-click the *ServiceSoapBinding* node to open the interface editor.
2. Open the **WSDL Content** tab. A WSDL file is an XML file, and it may be difficult to view and understand it. However, a WSDL file is a specification of a web service, and the better you understand it, the better you can work with the service. The SoapUI interface helps you view your WSDL file:



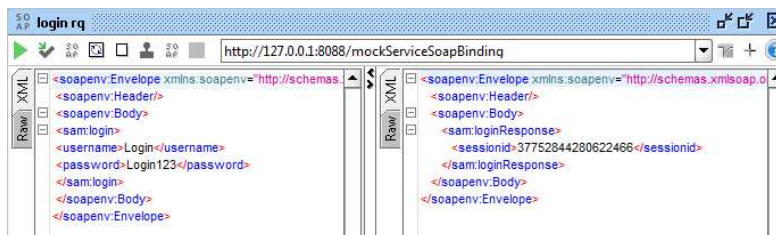
To learn more about WSDL files, see [Working with WSDLs](#).

Let's move on to web service requests:

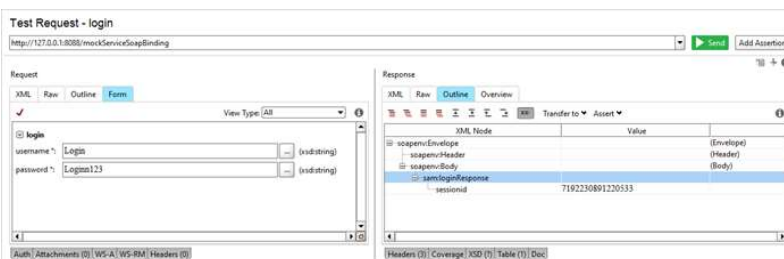
1. Expand the *login* node in the **Navigator** panel and double-click the *login rq* request. The request already contains the username and password.
2. Click ▶ to submit the request.

Note: Make sure you [run the mock service](#) to get a response.

You should now see the response in the Response panel:



Tip: ReadyAPI allows you to see requests and responses in different formats



Track Test Performance As You Scale Your API Testing



SoapUI

- ✓ Support for SOAP and REST API Testing.
- ✗ Easy multi-environment switching.
- ✗ Detailed test history and test comparison reporting.



ReadyAPI

- ✓ Support for SOAP, REST, and GraphQL API Testing.
- ✓ Easy multi-environment switching.
- ✓ Detailed test history and test comparison reporting.

Try ReadyAPI

2. Functional Tests of Web Services

Introduction

Let's look at how tests work in SoapUI. The sample project contains three different test suites all containing different test cases. A test case is made up of several test steps and load tests. So, the structure of a test suite looks like this:

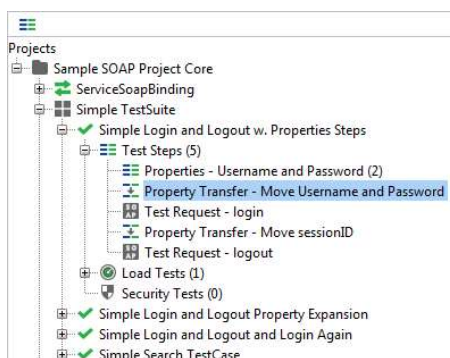
- Test suites
 - Test cases
 - Test steps
 - Load tests

Note: As you can see, there are also load tests. These will be described [later](#) in this tutorial.

Tutorial

Now let's examine a functional test in a test case.

1. Expand *Simple TestSuite* and double-click the first test case: *Simple Login and Logout w. Properties Steps*:



As you can see, the test case consists of five test steps.

2. Open the Description tab to see the test case description.

This one is easy...

The test will first login and then it will logout. :-)

We are using a Property Transfer Step to move password and user name from a Properties Step to the Request and a second Property Transfer Step to move the sessionID we got in the login response to the logout request..

Description Properties Setup Script TearDown Script

3. The step consists of test steps of three different types:

1. One *Properties* test step
2. Two *SOAP Requests* test steps
3. Two *Property Transfer* test steps.

This is what they do:

1. **Properties:**

Stores properties for later use. In our case, these are the *Username* and *Password* properties required to log in.

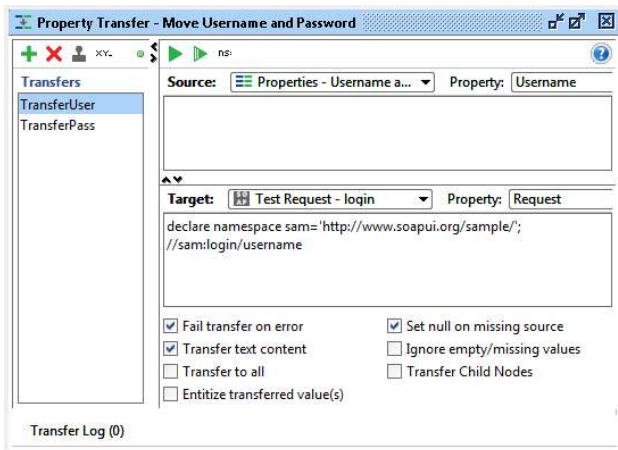
2. **SOAP Request:**

The actual requests to the server. There are two requests in this test case: *login* and *logout* requests.

3. **Property Transfer:**

A step is used for moving property values between different steps. You can use it to move values from the *Properties* test step, to a request like the *Property Transfer: Move Username and Password*. You can also move properties from a request test step to another request test step, like the *Property Transfer: Move sessionID* test step does.

4. Double-click the *Property Transfer: Move Username and Password* test step.



This property transfer test step takes values from source properties and assigns them to target properties. Examine the *TransferUser* transfer. It takes the *Username* property from the *Properties: Username and Password* test step and moves it to the *Test Request: login* test step. Then look at the *TransferPass* transfer. It takes the *Password* property from the *Properties: Username and Password* test step and moves it to the *Test Request: login* test step as well.

5. An important feature in SoapUI is *assertions*. Assertions validate that the response is what we expected. Open the *Test Request: login* test step.

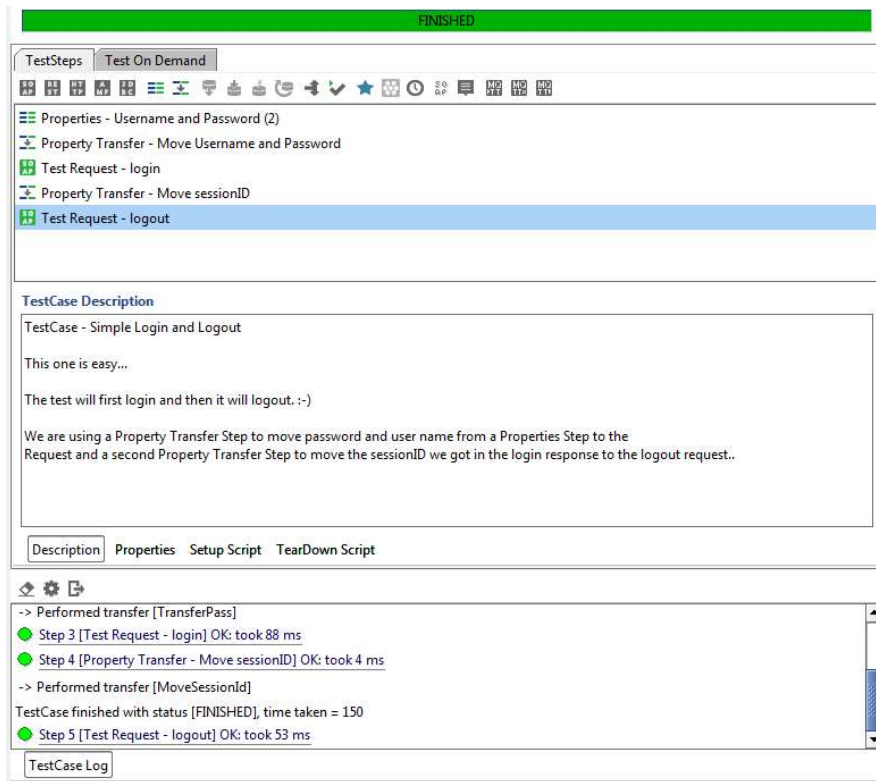
In the test step, we have three assertions that assert different things:

1. **SOAP Response** - Asserts that the response is a SOAP response.
2. **Schema Compliance** - Asserts that it complies with the Schema.
3. **Not SOAP Fault** - Asserts that it is not a SOAP fault.

To learn more about assertions, see [Getting Started With Assertions](#).

The Test Run

Now that we have examined the test case, let's run it! Press ► to run the test. The test case progress bar shows the progress:



If the test case failed:

- Does it turn red after only a step?

Don't forget to [start the mock service](#).

- Did you run the test step earlier?

Look at the responses - maybe a login request has already been sent.

If so, you logged in earlier and are not allowed to log in again. To fix that, go to the mock service, stop and restart the service. This will clear information about all the sessions running on the server. Now, rerun the test.

3. Web Service Load Tests

Introduction

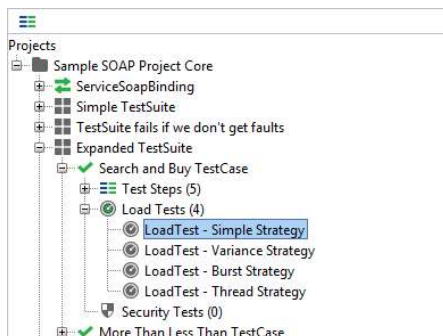
Another useful feature in SoapUI is the ability to quickly create load tests. Why is this useful? The earlier you are able to run load tests, the earlier you can discover any performance issues. In SoapUI, you create a load test by right-clicking a functional test and selecting **New LoadTest**. Done!

A SoapUI load test allows you to immediately test that the web service can respond quickly to the same request over and over again.

To learn more, see [Load Testing Overview](#).

Tutorial

1. Go to *Expanded TestSuite > Search and Buy TestCase*. There are four different load tests in that test case, one for each load test strategy.



For now, we will choose the *LoadTest: Simple Strategy* load test. This test is based on the *Simple* load strategy, which is a very basic strategy with a random delay. To learn more about strategies, see [Simulating Different Types of Load](#).


2. Let's configure the test.

Threads:	<div>1</div>	Strategy	<div>Simple</div>	Test Delay	<div>1000</div>	Random	<div>0.5</div>				
	Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
	Properties - Username and Pa...	0	0	0	0	0	0	0	0	0	0
	Test Request - login	0	0	0	0	0	0	0	0	0	0
	Test Request - search	0	0	0	0	0	0	0	0	0	0
	Test Request - Buy	0	0	0	0	0	0	0	0	0	0
	Test Request - logout	0	0	0	0	0	0	0	0	0	0
	TestCase:	0	0	0	0	0	0	0	0	0	0

There are the following settings for the Simple strategy:


1. **Limit** - the number of seconds you want the test to run.
2. **Threads** - the number of used threads. For now, we will use one thread.
3. **Test Delay** - sets a delay between each test case run (in milliseconds). In our test, we have set it to 200 milliseconds.
4. **Random** - sets how the Test Delay value should be changed at random. The value 0.5 means the Random value should be half of the Test Delay value. That is, the actual delay will be between 100 and 300 milliseconds.

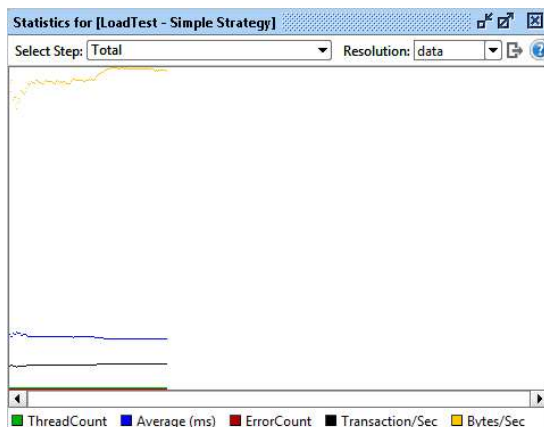
If you set Random to 0, there will be no random delay at all. In our case, it will be 200 milliseconds all the time.

3. Click  to run the load test.

As you can see, the table values are continuously updated. You can see values like the response times' throughput per second (tps), assertions, errors, percent of the test run, and more.

LoadTest - Simple Strategy										
Limit: 120 Seconds 53%										
Threads:	1	Strategy:	Simple	Test Delay:	200	Random:	0.5			
Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
Properties - Username and Pas...	0	0	0	0	304	0	0	0	0	0
Test Request - login	7	33	14.77	12	304	67.67	83392	18541	0	0
Test Request - search	8	442	18.62	15	304	53.7	386384	68252	0	0
Test Request - Buy	7	34	13.39	10	304	74.65	133760	32846	0	0
Test Request - logout	6	163	9.99	10	304	100.09	80864	26623	0	0
Test Case:	28	672	56.78	47	304	17.61	684400	39640	0	0

4. Click  to view the graph of the test run:



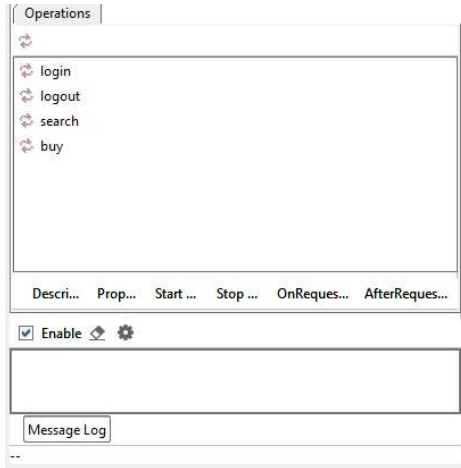
To learn more, see [Exporting Data and Statistics](#).

In SoapUI load tests, you can also do assertions. One of the most important assertions is the Max Errors assertion.

+ x ⚙			Online Help
Name	Step	Details	
Step Status	- Any -	testStep: - Any -, minRequests: 0, maxErrors: -1	
Max Errors	- Any -	testStep: - Any -, maxAbsoluteErrors: 100, maxRelativeErrors:...	

This assertion monitors the maximum number of errors allowed during a test run. Let's see how it works:

1. Double-click the ServiceSoapBinding mock service and click  to stop it.



2. Open the load test and click ► to run it. After a while, the test will fail. This will happen because the mock service is not available.

To learn about assertions in load tests, see [Validating Performance](#).

4. Web Service Mocking

Introduction

Web service mocking is a way to fake or simulate the functionality of a web service.

Web service mocking is very useful in projects where implementation of a web service has not started or is not finished yet, or where you, for some other reason, cannot access the web service.

In other words, web service mocking makes it possible for you to start creating a test for a web service at the same time you start developing the web service. This means that when the real web service is ready for testing, you already have the tests done. This can be extremely powerful and allows the use of test-driven and Agile methodologies in your workflow.

In SoapUI, you can create a mock service for a single web service request or generate a mock service containing each defined response in the interface.

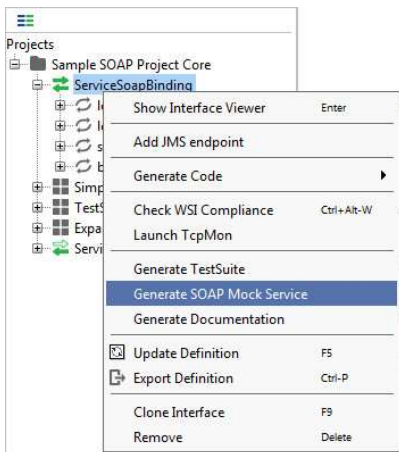
To learn more about mock services, see [Service Mocking Overview](#).

You may also be interested in ServiceV that provides more functionality for creating virtual services. Try [ReadyAPI](#) for free.

Tutorial

To add a mock service:

1. Right-click the interface and select **Generate SOAP Mock Service**.



2. In the **Generate MockService** dialog, you can specify the desired parameters for the new mock service. Leave the default settings for now and click **OK**.

Set options for generated MockOperations for this Interface

MockService: <create>

Operations:

☒ login
☒ logout
☒ search
☒ buy

Path: /mockServiceSoapBinding

Port: 8088

Add Endpoint: ☒ Adds the MockServices endpoint to the mocked Interface

Start MockService: ☐ Starts the MockService immediately

3. Enter the name of your mock service and click **OK**.

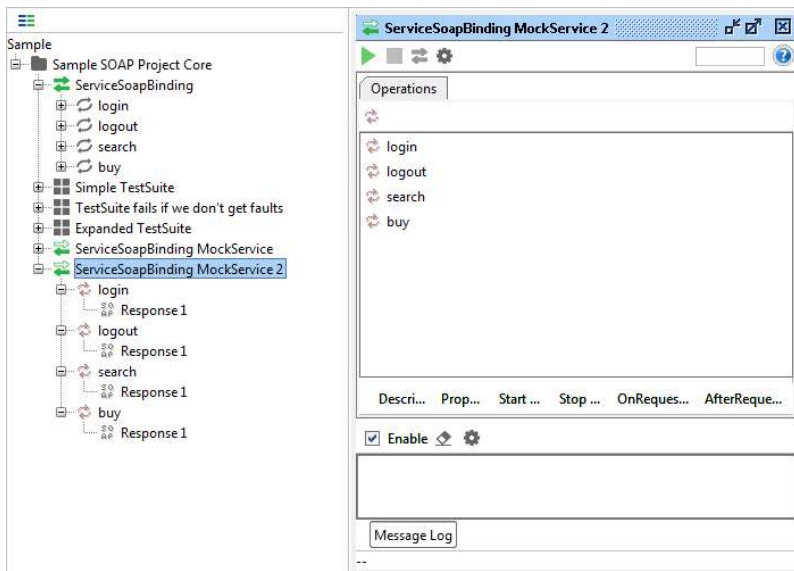
Generate SOAP Mock Service

Specify name of MockService to create

ServiceSoapBinding MockService

Note: The sample project already contains a mock service with the default name. Make sure you specify a different name for a new mock service.

The mock service will now be generated successfully:

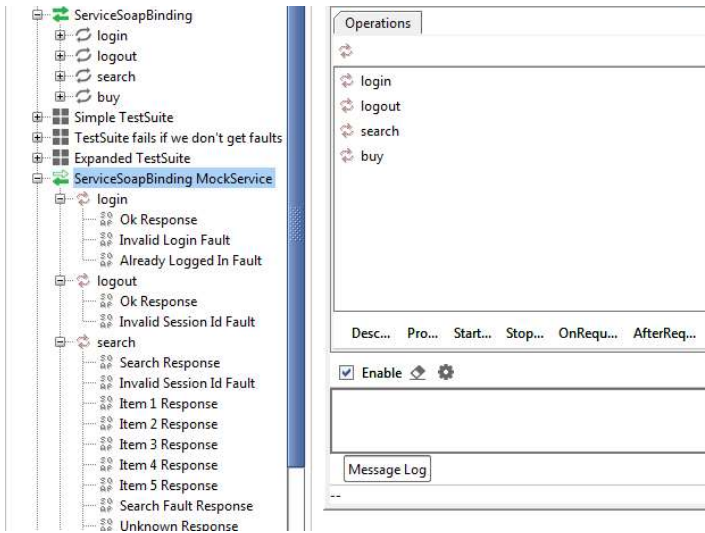


The screenshot shows the SoapUI interface. On the left, the 'Sample' project tree is expanded, showing the 'ServiceSoapBinding' interface with its operations (login, logout, search, buy) and their respective responses. The 'ServiceSoapBinding MockService 2' is highlighted. On the right, the 'ServiceSoapBinding MockService 2' configuration window is open, showing the 'Operations' tab with the same list of operations (login, logout, search, buy) and their responses. The 'Enable' checkbox is checked, and the 'Message Log' is visible at the bottom.

You can then choose how to respond to the request and choose how to dispatch (send out) different responses.

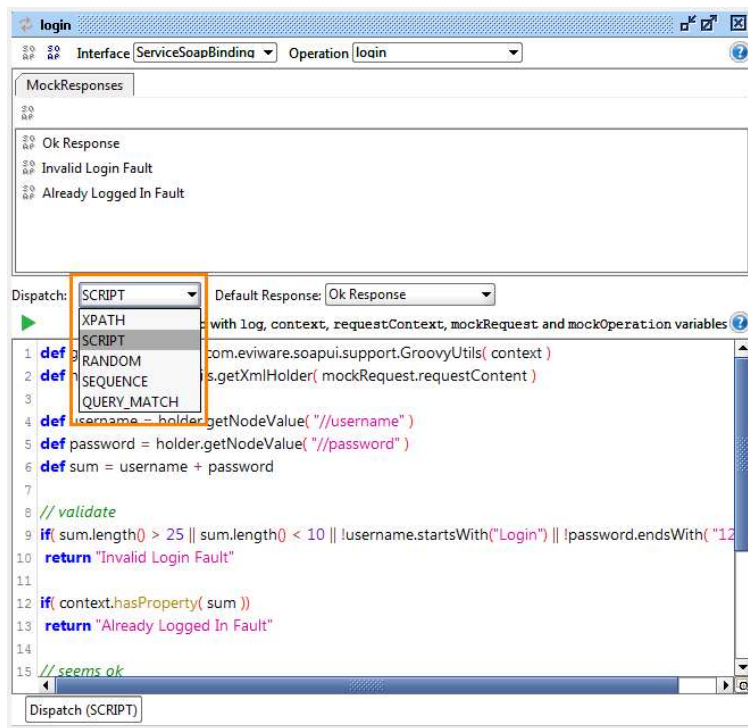
The sample project already contains the *ServiceSoapBinding* mock service. Let's take a look at how the mock service works:

1. Open it by double-clicking ServiceSoapBinding MockService:

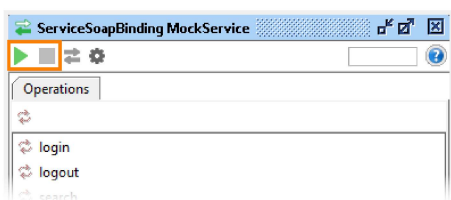


2. Browse through different operations in the mock service (login, logout, search, and buy) and see what we do with different requests. As you can see, all the requests are dispatched by using scripts. This is by far the most common way of dispatching, but early on in a project you might just want to create a set of responses and dispatch them in sequence or at random.

To change the dispatch method, use the Dispatch drop-down list:



3. To start and stop the mock service, use the buttons on the mock service toolbar:



Go ahead, try it!

Docs

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

More

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

Explore SmartBear Products 

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

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



© 2023 SmartBear Software. All Rights Reserved.

[Privacy](#) | [Terms of Use](#) | [Website Terms of Use](#) | [Security](#)

