**What is HTTP?**

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

what is json and how is the format look like?

JSON: **J**ava**S**cript **O**bject **N**otation.

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

## Exchanging Data

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

## Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

var myObj = { "name":"John", "age":31, "city":"New York" };  
var myJSON = JSON.stringify(myObj);  
window.location = "demo\_json.php?x=" + myJSON;

You will learn more about the JSON.stringify() function later in this tutorial.

## Receiving Data

If you receive data in JSON format, you can convert it into a JavaScript object:

### Example

var myJSON = '{ "name":"John", "age":31, "city":"New York" }';  
var myObj = JSON.parse(myJSON);  
document.getElementById("demo").innerHTML = myObj.name;

You will learn more about the JSON.parse() function later in this tutorial.

## Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

### Example

Storing data in local storage

//Storing data:  
myObj = { "name":"John", "age":31, "city":"New York" };  
myJSON = JSON.stringify(myObj);  
localStorage.setItem("testJSON", myJSON);  
  
//Retrieving data:  
text = localStorage.getItem("testJSON");  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;

## What is JSON?

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is "self-describing" and easy to understand
* JSON is language independent **\***

\*  
JSON uses JavaScript syntax, but the JSON format is text only.  
Text can be read and used as a data format by any programming language.

The JSON format was originally specified by [Douglas Crockford](http://www.crockford.com).

## Why use JSON?

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

**Two HTTP Request Methods: GET and POST**

Two commonly used methods for a request-response between a client and server are: GET and POST.

* **GET** - Requests data from a specified resource
* **POST** - Submits data to be processed to a specified resource

**The GET Method**

**Note that the query string (name/value pairs) is sent in the URL of a GET request:**

/test/demo\_form.php**?name1=value1&name2=value2**

**Some other notes on GET requests:**

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions
* GET requests should be used only to retrieve data

**The POST Method**

**Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:**

POST /test/demo\_form.php HTTP/1.1  
Host: w3schools.com  
**name1=value1&name2=value2**

**Some other notes on POST requests:**

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length

**Compare GET vs. POST**

The following table compares the two HTTP methods: GET and POST.

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL  Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

**Other HTTP Request Methods**

The following table lists some other HTTP request methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| HEAD | Same as GET but returns only HTTP headers and no document body |
| PUT | Uploads a representation of the specified URI |
| DELETE | Deletes the specified resource |
| OPTIONS | Returns the HTTP methods that the server supports |
| CONNECT | Converts the request connection to a transparent TCP/IP tunnel |

Validating with JSON Schema

The simplest way to check if JSON is valid is to load the JSON into a JObject or JArray and then use the [IsValid(JToken, JsonSchema)](http://www.newtonsoft.com/json/help/html/M_Newtonsoft_Json_Schema_Extensions_IsValid.htm) method with the JSON Schema.

**Validate JSON with IsValid**

[Copy](http://www.newtonsoft.com/json/help/html/JsonSchema.htm)

1string schemaJson = @"{

2 'description': 'A person',

3 'type': 'object',

4 'properties':

5 {

6 'name': {'type':'string'},

7 'hobbies': {

8 'type': 'array',

9 'items': {'type':'string'}

10 }

11 }

12}";

13

14JsonSchema schema = JsonSchema.Parse(schemaJson);

15

16JObject person = JObject.Parse(@"{

17 'name': 'James',

18 'hobbies': ['.NET', 'Blogging', 'Reading', 'Xbox', 'LOLCATS']

19}");

20

21bool valid = person.IsValid(schema);

22// true

To get validation error messages, use the [IsValid(JToken, JsonSchema, IList< String>)](http://www.newtonsoft.com/json/help/html/M_Newtonsoft_Json_Schema_Extensions_IsValid_1.htm) or [Validate(JToken, JsonSchema, ValidationEventHandler)](http://www.newtonsoft.com/json/help/html/M_Newtonsoft_Json_Schema_Extensions_Validate_1.htm) overloads.

**Validate JSON with IsValid**

[Copy](http://www.newtonsoft.com/json/help/html/JsonSchema.htm)

1JsonSchema schema = JsonSchema.Parse(schemaJson);

2

3JObject person = JObject.Parse(@"{

4 'name': null,

5 'hobbies': ['Invalid content', 0.123456789]

6}");

7

8IList<string> messages;

9bool valid = person.IsValid(schema, out messages);

10// false

11// Invalid type. Expected String but got Null. Line 2, position 21.

12// Invalid type. Expected String but got Float. Line 3, position 51.

Internally IsValid uses [JsonValidatingReader](http://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonValidatingReader.htm) to perform the JSON Schema validation. To skip the overhead of loading JSON into a JObject/JArray, validating the JSON, and then deserializing the JSON into a class, JsonValidatingReader can be used with JsonSerializer to validate JSON while the object is being deserialized.

**Validate JSON with JsonValidatingReader**

[Copy](http://www.newtonsoft.com/json/help/html/JsonSchema.htm)

1string json = @"{

2 'name': 'James',

3 'hobbies': ['.NET', 'Blogging', 'Reading', 'Xbox', 'LOLCATS']

4}";

5

6JsonTextReader reader = new JsonTextReader(new StringReader(json));

7

8JsonValidatingReader validatingReader = new JsonValidatingReader(reader);

9validatingReader.Schema = JsonSchema.Parse(schemaJson);

10

11IList<string> messages = new List<string>();

12validatingReader.ValidationEventHandler += (o, a) => messages.Add(a.Message);

13

14JsonSerializer serializer = new JsonSerializer();

15Person p = serializer.Deserialize<Person>(validatingReader);

Creating JSON Schemas

The simplest way to get a [JsonSchema](http://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Schema_JsonSchema.htm) object is to load it from a string or a file.

**Creating JSON Schemas from strings or files**

[Copy](http://www.newtonsoft.com/json/help/html/JsonSchema.htm)

1// load from a string

2JsonSchema schema1 = JsonSchema.Parse(@"{'type':'object'}");

3

4// load from a file

5using (TextReader reader = File.OpenText(@"c:\schema\Person.json"))

6{

7 JsonSchema schema2 = JsonSchema.Read(new JsonTextReader(reader));

8

9 // do stuff

10}

It is also possible to create JsonSchema objects in code.

**Create new JSON Schemas in code**

[Copy](http://www.newtonsoft.com/json/help/html/JsonSchema.htm)

1JsonSchema schema = new JsonSchema();

2schema.Type = JsonSchemaType.Object;

3schema.Properties = new Dictionary<string, JsonSchema>

4{

5 { "name", new JsonSchema { Type = JsonSchemaType.String } },

6 {

7 "hobbies", new JsonSchema

8 {

9 Type = JsonSchemaType.Array,

10 Items = new List<JsonSchema> { new JsonSchema { Type = JsonSchemaType.String } }

11 }

12 },

13};

14

15JObject person = JObject.Parse(@"{

16 'name': 'James',

17 'hobbies': ['.NET', 'Blogging', 'Reading', 'Xbox', 'LOLCATS']

18}");

19

20bool valid = person.IsValid(schema);

21// true

Difference between SOAP and REST

 There are many differences between SOAP and REST web services. The important 10 differences between SOAP and REST are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards** to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

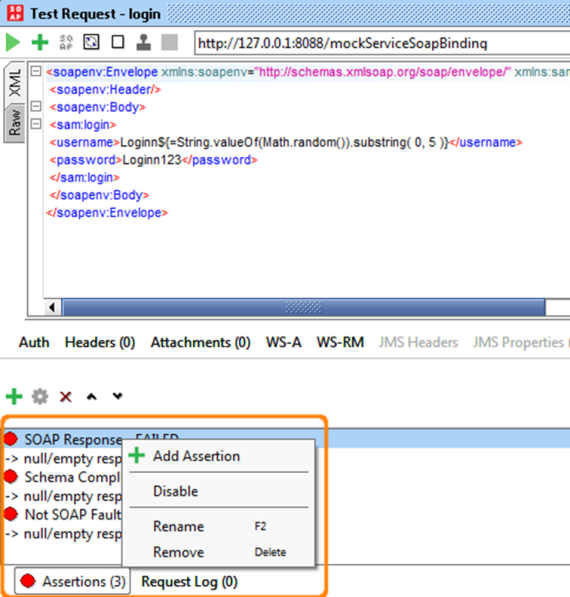
Assertions in soapUI

Assertions are used to validate the message received by a TestStep during execution, usually by comparing parts of the message (or the entire message) to some expected value. Any number of assertions can be added to a sampler TestStep, each validating some different aspect or content of the response. After a sampler TestStep executes, all its assertions are applied to the received response and if any of them fail the TestStep is marked as failed in the TestCase view and a corresponding FAILED entry is shown in the Test Execution Log.

Here we can see that the "Test Request - login" TestStep has failed, which in the TestCase Run Log at the bottom also displays details on the actual assertion failure; "took 1023 ms" means that the "SLA" assertion failed, i.e. the response was too slow.

### 1. Managing Assertions

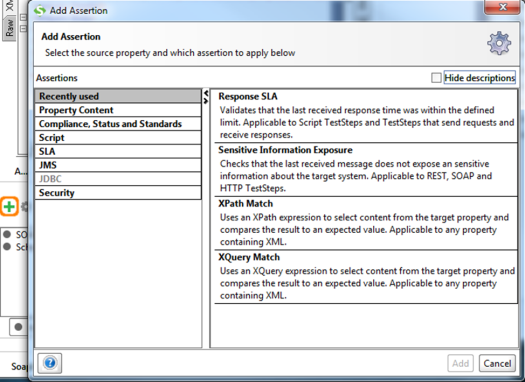
Assertions are always displayed in a tab at the bottom of the containing TestSteps' editor window.



In the screenshot above you see the 3 assertions added to a SOAP Request TestStep and they all failed.

The toolbar on top of the assertions allows you to add, configure, delete, move and clone assertions as required and the right-click popup menu for an assertion contains similar actions  (as can be seen in the screenshot). Double-clicking an assertion brings up its configuration dialog if available.

Adding an assertion is straight forward; press the left-most button in the toolbar, which opens a dialog which prompts you to choose which assertion to add.



The dialog will contain map of assertions available for the current type of sampler (see below), pressing OK will add the assertion and open its configuration dialog.

### 2. Assertion categories

Assertions are divided in several categories for easier managing. Only categories that contain applicable Assertions for specific type of sampler are enabled. In case of AssertionTestStep only categories that contain applicable Assertions for selected Source/Property combination are enabled ( please check [http://www.soapui.org/Functional-Testing/assertion-test-step.html](https://www.soapui.org/Functional-Testing/assertion-test-step.html) ). Same goes for enabling/disabling particular Assertion inside the category.

In addition under the Categories there is "Recently used" category which lists 5 Assertions most recently used, some of which may also be disabled for particular Source/Property selected.

All available Categories with containing Assertions are listed below.

#### 2.1. Property Content Category

* Contains - Searches for the existence of a string token in the property value, supports regular expressions. Applicable for any property.
* Message Content Assertion - Allows for complex content validation of XML messages. Applicable to any property containing XML.
* Not Contains - Searches for the non-existence of a string token in the property value, supports regular expressions. Applicable to any property.
* XPath Match - uses an XPath expression to select content from the target property and compares the result to an expected value. Applicable to any property containing XML.
* XQuery Math - uses an XQuery expression to select content from the target property and compares the result to an expected value. Applicable to any property containing XML.

#### 2.1.1. Compliance, Status and Standards

* HTTP Download all resource - downloads all resources referred to be an HTML document (images, scripts, etc.) and validates that they are all available. Applicable to any property containing HTML.
* Invalid HTTP Status Codes - checks that the target TestStep received an HTTP result with a status code not in the list of defined codes. Applicable to any TestStep that receives HTTP messages
* Not SOAP Fault - validates that the last received message is not a SOAP Fault. Applicable to SOAP TestSteps.
* Schema Compliance - validates that the last received message is compliant with the associated WSDL or WADL schema definition. Applicable to SOAP and REST TestSteps. The schema definition URL supports [Property Expansions](https://www.soapui.org/scripting---properties/property-expansion.html) (e.g. ${#System#my.wsdl.endpoint}/services/PortType?wsdl ).
* SOAP Fault - validates that the last received message is a SOAP Fault. Applicable to SOAP TestSteps SOAP Request - validates that the last received request is a valid SOAP Request. Applicable to MockResponse TestSteps only.
* SOAP Response - validates that the last received response is a valid SOAP Response. Applicable to SOAP TestRequest Steps only.
* Valid HTTP Status Codes - checks that the target TestStep received an HTTP result with a status code in the list of defined codes. Applicable to any TestStep that receives HTTP messages.
* WS-Addressing Request - validates that the last received request contains valid WS-Addressing Headers. Applicable to MockResponse TestSteps only.
* WS-Addressing Response - validates that the last received response contains valid WS-Addressing Headers. Applicable to SOAP TestRequest Steps only.
* WS-Security Status - validates that the last received message contained valid WS-Security headers. Applicable to SOAP TestSteps.

#### 2.1.1.1. Script

* Script Assertion - runs a custom script to perform arbitrary validations. Applicable to TestSteps only (i.e. not properties).

#### 2.1.1.2. SLA

* Response SLA - validates that the last received response time was within the defined limit. Applicable to Script TestSteps and TestSteps that send requests and receive responses.

#### 2.1.1.3. JMS

* JMS Status - validates that the JMS request of the target TestStep executed successfully. Applicable to Request TestSteps with a JMS endpoint.
* JMS Timeout - validates that the JMS statement of the target TestStep did not take longer than the specified duration. Applicable to Request TestSteps with a JMS endpoint.

#### 2.1.1.4. JDBC

* JDBC Status - validates that the JDBC statement of the target TestStep executed successfully. Applicable to JDBC TestSteps only.
* JDBC Timeout - validates that the JDBC statement of the target TestStep did not take longer than the specified duration. Applicable to JDBC TestSteps only.

#### 2.1.1.5. Security

* Sensitive Information Exposure - Checks that the last received message does not expose an sensitive information about the target system. Applicable to REST, SOAP and HTTP TestSteps.

### 3. Common Assertions

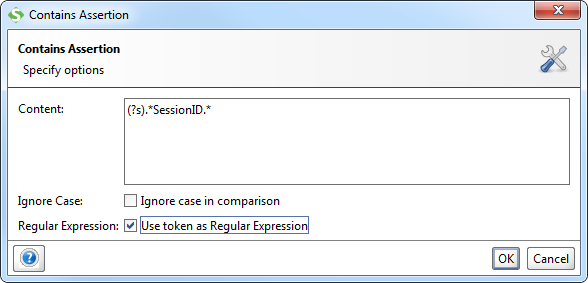
A large number of assertions are available for validating received messages, some being specific for a certain type of TestStep and some being common for all TestSteps. The assertions available for all TestSteps are:

* Contains - checks for the existence of a specified string (see below).
* Not Contains - checks for the non-existence of a specified string (see below).
* Reponse SLA - check the response time against a specified value (see below).
* XPath Match - compares the result of an XPath expression to an expected value ([read more](https://www.soapui.org/functional-testing/validating-messages/validating-xml-messages.html)).
* XQuery match - compares the result on an XQuery expression to an expected value ([read more](https://www.soapui.org/functional-testing/validating-messages/validating-xml-messages.html)).
* Script - runs an arbitrary script that can be used to validate the received message as desired ([read more](https://www.soapui.org/functional-testing/validating-messages/using-script-assertions.html)).

The first three of these are covered in more detail below.

### 3.1. The Contains Assertion

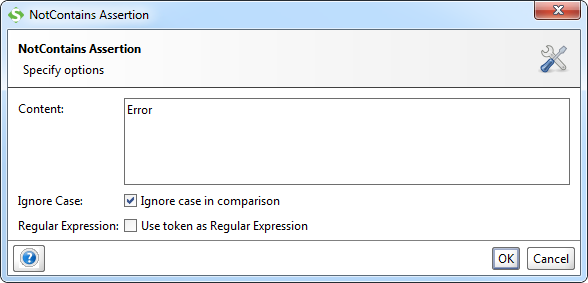
This assertion checks for the existence of some text in the received response or request message, its configuration dialog is as follows:



The example shown in the screenshot specifies a reqular expression checking for the string "SessionID" in the entire content of the message to be validated, note the "(?s)" construct to specify whitespace handline ([supported RegEx format](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html)). Property-Expansion is supported in the Content field.

### 3.2. The Not Contains Assertion

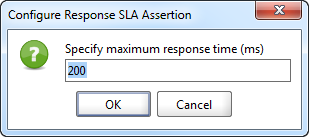
The counterpart for the Contains assertion; this one checks that a specified spring does not exist in the asserted message. The configuration dialog is the same as above:



Here the example in the screenshot just checks for the non-existence of the token "Error" in the entire response.

### 3.3. The Response SLA Assertion

This assertion validates that the response time is within the specified value, otherwise the assertion will fail. The configuration dialog is a simple one:



Property-Expansion is supported in the specified value, allowing you to control the assertion limit via some external mechanism if needed.

### 4. TestStep specific Assertions

A number of assertions are available that are specific to one or several types of TestSteps. These are covered in more detail together with the corresponding TestStep but also listed below to give you an overview:

For SOAP Request TestSteps the following assertions are available (see [Validating SOAP Messages](https://www.soapui.org/soap-and-wsdl/validating-soap-services.html) for more details):

* Schema Compliance - validates the response message against the definition in the WSDL and contained XML Schema.
* SOAP Response - checks that the response is a valid SOAP Response.
* SOAP Fault - checks that the response is a SOAP Fault.
* Not SOAP Fault - checks that the response is not a SOAP Fault.
* WS-Security Status - validates the WS-Security headers and tokens in the response.
* WS-Addressing Response - validates the WS-Addressing Headers in the response.
* JMS Timeout - when using a JMS endpoint that specifies a response channel this assertion validates that the response is received within the specified time.
* JMS Status - when using a JMS endpoint, this assertion validates that no JMS-related errors occured.

For REST Request TestSteps the following assertions are also available:

* Schema Compliance - validates the response message against the schema that has either been specified in an underlying WADL file or that has been inferred from the response.

For JDBC Request TestSteps the following assertions are also available:

* JDBC Timeout - validates that the result is received from the target database within the specified time.
* JDBC Status - validates that no JDBC-related errors occured.

For MockResponse TestSteps the following assertions are available:

* Schema Compliance - validates the request message against the definition in the WSDL and contained XML Schema.
* SOAP Request - checks that the request is a valid SOAP Request.
* WS-Security Status - validates the WS-Security headers and tokens in the request.
* WS-Addressing Request - validates the WS-Addressing Headers in the request.

**What is XPath?**

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.

* XPath is a syntax for defining parts of an XML document
* XPath uses path expressions to navigate in XML documents
* XPath contains a library of standard functions
* XPath is a major element in XSLT and in XQuery

XPath is a W3C recommendation

|  |  |
| --- | --- |
|  |  |

**XPath Path Expressions**

XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages

**XPath is Used in XSLT**

XPath is a major element in the XSLT standard.

With XPath knowledge you will be able to take great advantage of XSL.

**XPath Example**

We will use the following XML document:

<?xml version="1.0" encoding="UTF-8"?>  
  
<bookstore>  
  
<book category="cooking">  
  <title lang="en">Everyday Italian</title>  
  <author>Giada De Laurentiis</author>  
  <year>2005</year>  
  <price>30.00</price>  
</book>  
  
<book category="children">  
  <title lang="en">Harry Potter</title>  
  <author>J K. Rowling</author>  
  <year>2005</year>  
  <price>29.99</price>  
</book>  
  
<book category="web">  
  <title lang="en">XQuery Kick Start</title>  
  <author>James McGovern</author>  
  <author>Per Bothner</author>  
  <author>Kurt Cagle</author>  
  <author>James Linn</author>  
  <author>Vaidyanathan Nagarajan</author>  
  <year>2003</year>  
  <price>49.99</price>  
</book>  
  
<book category="web">  
  <title lang="en">Learning XML</title>  
  <author>Erik T. Ray</author>  
  <year>2003</year>  
  <price>39.95</price>  
</book>  
  
</bookstore>

In the table below we have listed some XPath expressions and the result of the expressions:

|  |  |
| --- | --- |
| **XPath Expression** | **Result** |
| /bookstore/book[1] | Selects the first book element that is the child of the bookstore element |
| /bookstore/book[last()] | Selects the last book element that is the child of the bookstore element |
| /bookstore/book[last()-1] | Selects the last but one book element that is the child of the bookstore element |
| /bookstore/book[position()<3] | Selects the first two book elements that are children of the bookstore element |
| //title[@lang] | Selects all the title elements that have an attribute named lang |
| //title[@lang='en'] | Selects all the title elements that have a "lang" attribute with a value of "en" |
| /bookstore/book[price>35.00] | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| /bookstore/book[price>35.00]/title | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

### **Difference between / and // Search Directives in XML**

/ Search directives :

1. It's starts search selection from root element in document.
2. XPath expressions is like absolute path from the root element.
3. /empinfo : When /empinfo is a absolute path from the root element. pretend as a root node.

You can use XPath search directives in middle of an XPath address to represent ancestor descendant relationships.

XPath addresses /empinfo/employee identifies employee elements. employee that is a next child (descendant) of empinfo element.

/empinfo/employee

<?xml version="1.0" standalone="yes"?>

<empinfo>

<employee id="1">

<name>Opal Kole</name>

<designation discipline="web">Senior Engineer</designation>

<email>OpalKole@myemail.com</email>

</employee>

<employee id="2">

<name from="CA">Max Miller</name>

<designation discipline="DBA">DBA Engineer</designation>

<email>maxmiller@email.com</email>

</employee>

</empinfo>

XPath expression select 3 to 12 line number employee node.

// Search directives :

1. It's starts search selection anywhere in XML document.
2. XPath expressions is like relative path from the context node.
3. //empinfo : When //empinfo define node path in anywhere in XML document. "//" find empinfo element locate any depth of XML document.

XPath addresses /empinfo//designation identifies any descendant designation element of element empinfo.

/empinfo//designation

<?xml version="1.0" standalone="yes"?>

<empinfo>

<employee id="1">

<name>Opal Kole</name>

<designation discipline="web">Senior Engineer</designation>

<email>OpalKole@myemail.com</email>

</employee>

<employee id="2">

<name from="CA">Max Miller</name>

<designation discipline="DBA">DBA Engineer</designation>

<email>maxmiller@email.com</email>

</employee>

</empinfo>

XPath expression select 5 and 10 line number designation node.

".//" (dot double forward slash) search directives identifies only descendant(child element) element. It's search relative path of current element.

.//name

<?xml version="1.0" standalone="yes"?>

<empinfo>

<employee id="1">

<name>Opal Kole</name>

<designation discipline="web">Senior Engineer</designation>

<email>OpalKole@myemail.com</email>

</employee>

<employee id="2">

<name from="CA">Max Miller</name>

<designation discipline="DBA">DBA Engineer</designation>

<email>maxmiller@email.com</email>

</employee>

</empinfo>

how to read attributes in xpath?

1. <books category="science">

2. <book id="7890">

3. <title>A New Kind of Science</title>

4. <edition>2</edition>

5. <author id="1234">Stephen Wolphram</author>

6. <prices>

7. <price currency="usd">20.00</price>

8. <price currency="can">25.00</price>

9. </prices>

10. </book>

11. <book id="8901">

12. <title>How To Build Your Own Neural Network</title>

13. <edition>4</edition>

14. <author id="6789">John Doe</author>

15. <prices>

16. <price currency="usd">24.00</price>

17. <price currency="can">28.00</price>

18. </prices>

19. </book>

20. </books>

Imagine that you wanted to obtain the ID for the book titled "A New Kind of Science".  Note that the book and author elements both come with an "id" attribute.  To get the ID for the book, you'd do this:

1. /books/book[title = "A New Kind of Science"]/@id

Which would return the attribute as follows:

1. id="7890"

Now, if you were interested in the value only, you could do this instead:

1. string(/books/book[title = "A New Kind of Science"]/@id)

Which would return the value as follows:

1. 7890

what is web service?

* A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.
* Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
* Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

To summarize, a complete web service is, therefore, any service that:

* Is available over the Internet or private (intranet) networks
* Uses a standardized XML messaging system
* Is not tied to any one operating system or programming language
* Is self-describing via a common XML grammar
* Is discoverable via a simple find mechanism

**Components of Web Services**

The basic web services platform is XML + HTTP. All the standard web services work using the following components

* SOAP (Simple Object Access Protocol)
* UDDI (Universal Description, Discovery and Integration)
* WSDL (Web Services Description Language)

what is XML and what will be format?

XML is a software- and hardware-independent tool for storing and transporting data.

## What is XML?

* XML stands for eXtensible Markup Language
* XML is a markup language much like HTML
* XML was designed to store and transport data
* XML was designed to be self-descriptive
* XML is a W3C Recommendation

## XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Tove from Jani, stored as XML:

<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>

The XML above is quite self-descriptive:

* It has sender information.
* It has receiver information
* It has a heading
* It has a message body.

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it:

## Note

To: Tove

From: Jani

### Reminder

Don't forget me this weekend!

## The Difference Between XML and HTML

XML and HTML were designed with different goals:

* XML was designed to carry data - with focus on what data is
* HTML was designed to display data - with focus on how data looks
* XML tags are not predefined like HTML tags are

## XML Does Not Use Predefined Tags

The XML language has no predefined tags.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

## XML is Extensible

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <data>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed, older version of the application can still work:

<note>  
  <date>2015-09-01</date>  
  <hour>08:30</hour>  
  <to>Tove</to>  
  <from>Jani</from>  
  <body>Don't forget me this weekend!</body>  
</note>

### Old Version

## Note

To: Tove

From: Jani

### Head: (none)

Don't forget me this weekend!

### New Version

## Note

To: Tove

From: Jani

Date: 2015-09-01 08:30

Don't forget me this weekend!

## XML Simplifies Things

* It simplifies data sharing
* It simplifies data transport
* It simplifies platform changes
* It simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

## XML is a W3C Recommendation

XML became a W3C Recommendation as early as in February 1998.

**steps to test soap and rest web services using soapUI pro**

## Ans: Why use SoapUI Pro for Web Service Testing?

While the Open Source version can be seen as the Swiss-Army knife for testing, SoapUI NG Pro is the tool with the sharpest edge. Applied to testing web services, SoapUI NG Pro focus on enhancing efficiency and usability. With [Point and Click Testing](https://www.soapui.org/Go-Pro/12-reasons-to-go-pro.html), you can drag and drop instead of manually writing the code. [The Form Editor](https://www.soapui.org/Go-Pro/12-reasons-to-go-pro.html) creates a form from your request, further eliminating the need for you to spend time on repetitive coding. These functions make, in combination with [The Outline Editor](https://www.soapui.org/Go-Pro/12-reasons-to-go-pro.html) who simplifies and exposes the XML structure, your testing more fun and less time consuming. If creativity, flow and speed are important to you as a tester, SoapUI NG Pro is too.

[Try the world's most powerful web services testing tool](https://www.soapui.org/downloads/download-soapui-pro-trial.html)

## Web Service Sample Project

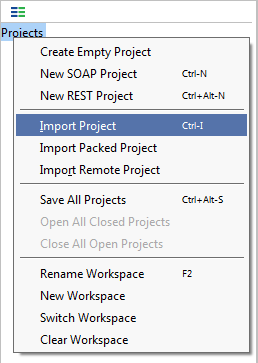
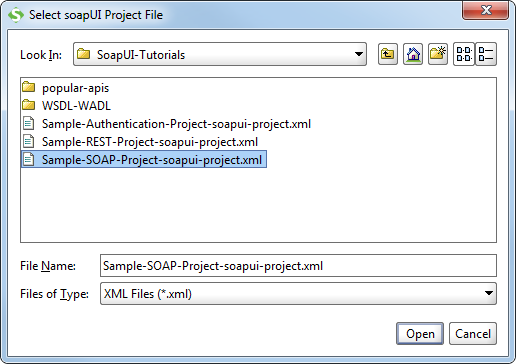
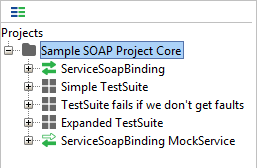
Getting started with a new application is always a challenge, no matter how complex it is. In order to make getting started with SoapUI easier for you, SoapUI comes with a sample SoapUI Project file that can be used as reference and inspiration.

It illustrates some of the basic concepts of SoapUI and can be used as a SoapUI starting point; try out the project, run the included MockService and the included tests to familiarize yourself with the s SoapUI interface, and then move on to the Your First SoapUI Project tutorial.

Here are the main concepts illustrated in the SoapUI Project.

1. Web Service Mocking
2. Web Service Inspection
3. Functional Tests of Web Services
4. Web Service Load Tests

Let start by opening a project.

1. Right click on Project Node in the navigator and select Import Project  
     
   A Select SoapUI Project dialog will open.
2. Select the project file sample-soapui-pro-project.xml or sample-soapui-project.xml from the tutorial folder User/soapUI-Tutorials (Note that you won't have this folder in case you unchecked the option during the installation).  
   
3. The Sample Project will be shown in the SoapUI Navigator.  
   

Now that we have the project loaded; let's move to Web Service Mocking!

## 1) [Web Service Mocking](https://www.soapui.org/downloads/servicev-pro.html)

### Introduction

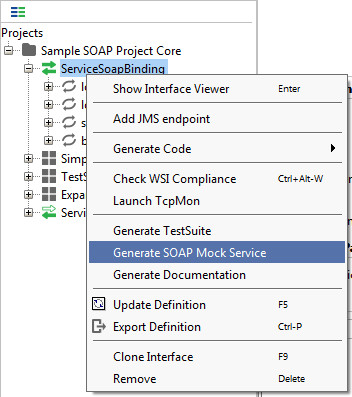
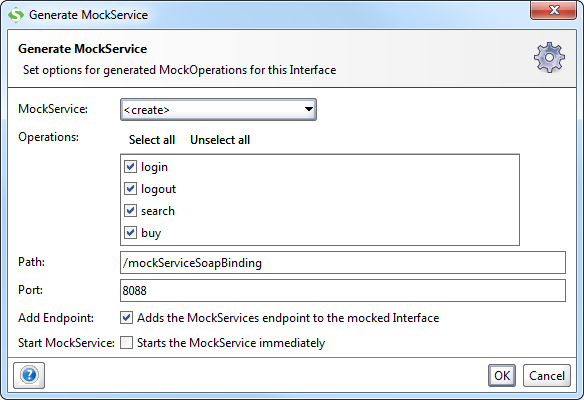
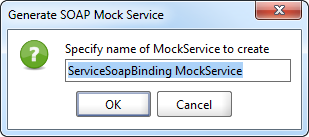
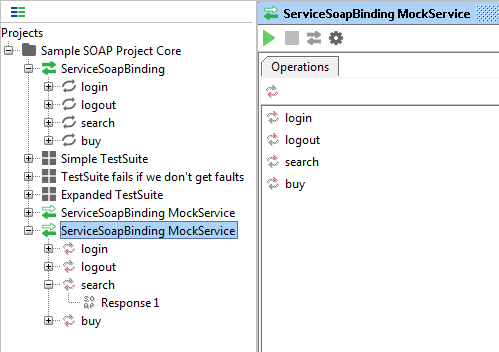
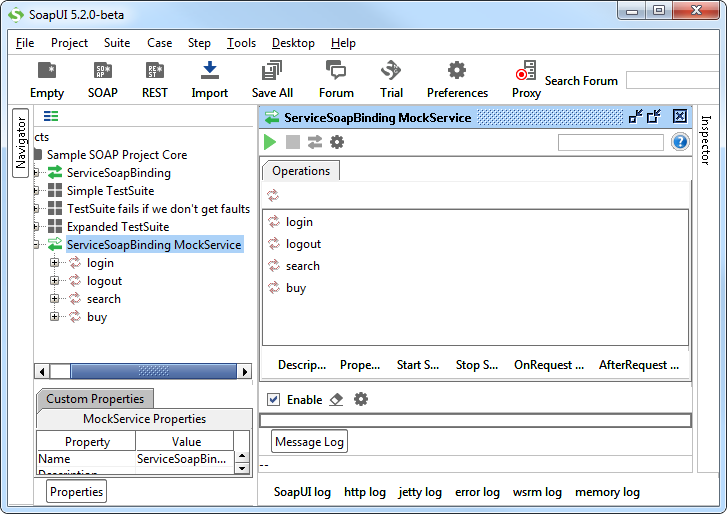
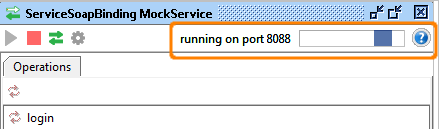
Web Service Mocking is way to fake or simulate the functionality of a Web Service in SoapUI; to create a Mock Up. But why do you need such functionality?

Web Service Mocking is very useful in projects where implementation of the Web Service has not started or is not finished for use yet or where you for some other reasons can't access the Web Services live.

In other words, Web Service Mocking makes it possible for you to start creating test for a Web Service the same time you start to develop the same Web Service. This means, when the real Web Service is ready for testing; you can already have the tests done. This can be extremely powerful and will make it possible to work according to Test Driven or Agile Methodologies.

In SoapUI you can create a MockService simply by choosing to add a Web Service Request to a MockService in order for the MockService to listen to it. Or you can even right-click the Interface Icon and choose "Generate MockService", which will generate a MockService containing each defined response in the Interface.

### Tutorial

1. To add a MockService, right click on the interface icon and select Generate MockService.  
   
2. The Generate MockService dialog will open. 
3. Enter the name of your MockService and click OK.  
     
   **Note:** You will receive a Composite Warning if you have two mockservices with the same name.
4. The MockService will now be successfully generated.  
     
   You can then choose how to Respond to the Request and choose how to Dispatch (send out) different responses.
5. Now, let's take a look at how the Mock Service works. There is a SampleServiceSoapBinding Mock Service already in the Sample Pro Project, so if you followed the steps above, please discard what you've created or move to the SampleServiceSoapBinding MockService. Open the MockService by double clicking on it.  
   
6. Browse through the different Interfaces in the MockService; login, logout, search, and buy, and see what we do with different requests. As you see all the requests are dispatched 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.
7. Click on the Run button runto start the MockService.
8. You should now see the MockService running on port 8088.  
   

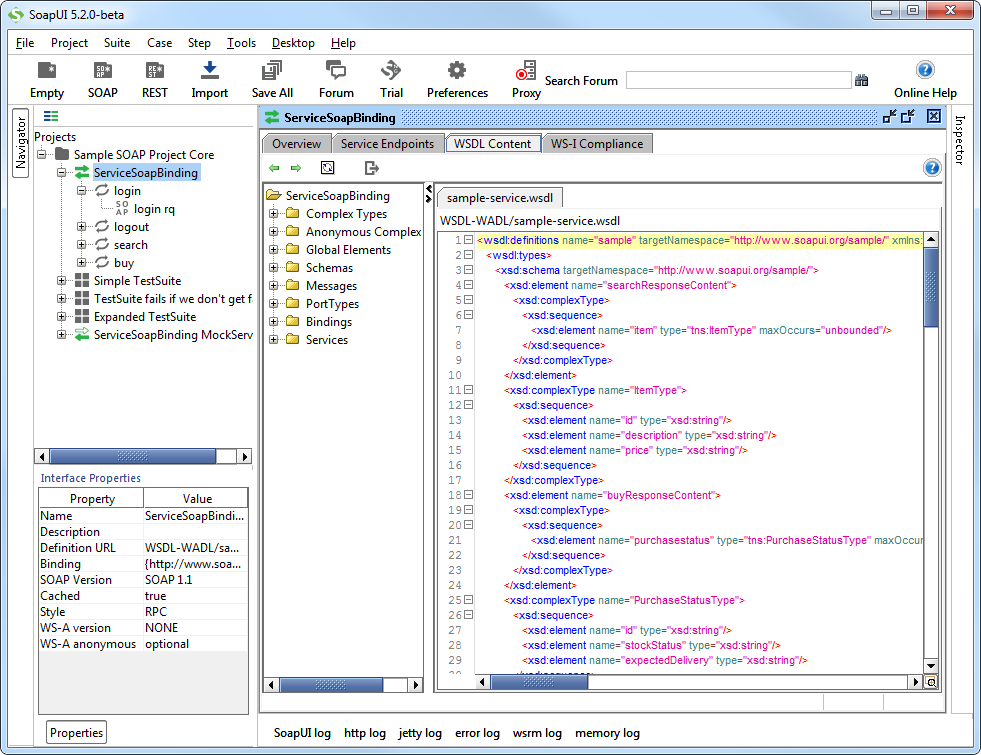
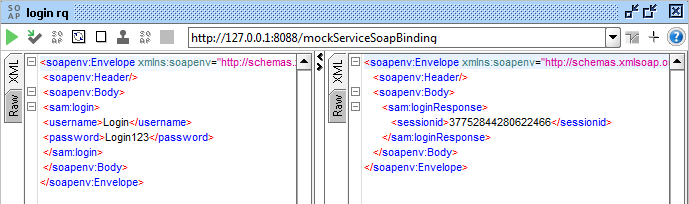
## 2) Web Service Inspection

### Introduction

Web Service Inspection a fancy way of saying looking at the Web Service. Good support for Web Service inspection can be very important early in your testing process when you want to find out how a Web Service works.

Inspecting the Web Service can be done in two ways; inspecting the Web Service and making Web Service requests.

### Tutorial

1. Let's start by some Inspection. Go to the Interface Inspector by double clicking the SampleServiceSoapBinding node. You will see the Interface Inspector open on the SoapUI desktop.
2. This is a structured way of looking at your WSDL Just browsing the XML of the WSDL is very cluttered and a WSDL often is to complicated too look at in its entirety and therefore it is seldom done, resulting in lower WSDL understanding. In other words, since WSDL's are so complicated people don't read it, and remember, the WSDL is the specification (contract actually, bet let's not get into that...) and how well you use it depends on how well you understand it. This is a serious problem early on in projects, the people in the project don't understand the Contract. The SoapUI Interface Viewer is a great tool for solving this problem.  
     
   
3. Expand the "login" and double click on Request1. You will see loginRequest open on the soapUI desktop. The request already uses the username Login and the password Login123
4. Click on the Run button runto submit the request.
5. You should now see response on the response panel  
   

**Pro:**

You can click on the different tabs to view response in different format.

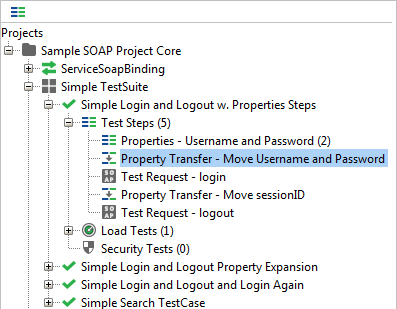
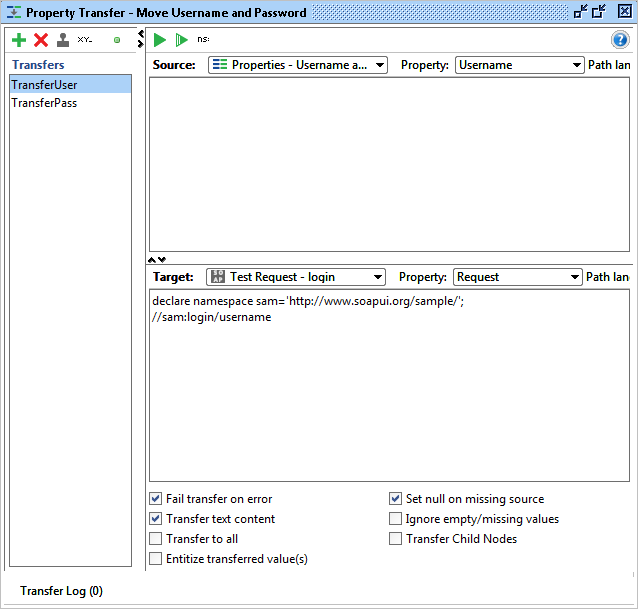
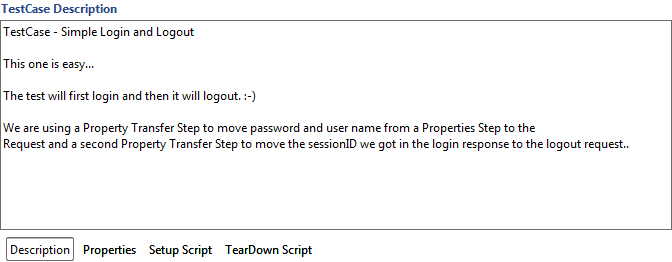
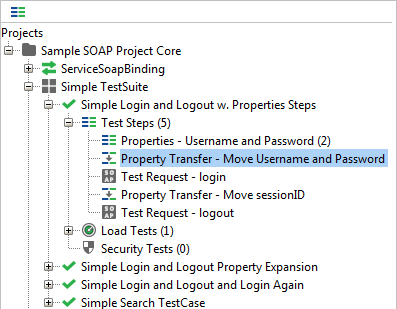
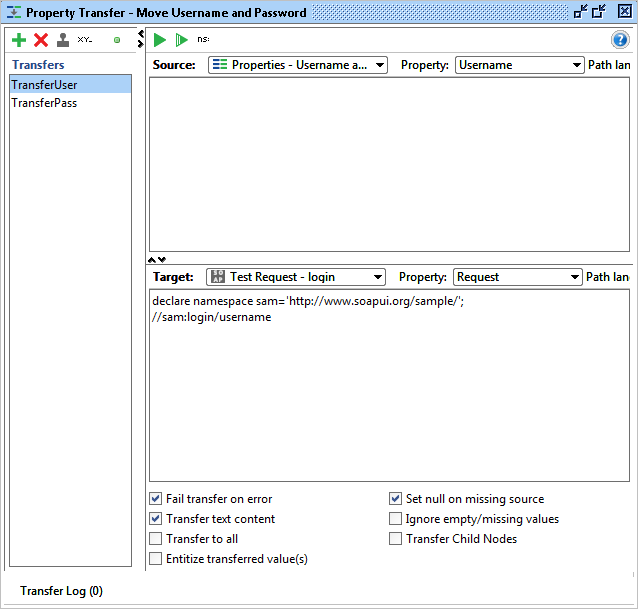
## 3) Functional Tests of Web Services

### Introduction

Let's move on and look at how tests work in SoapUI. We have created three different TestSuites all containing different TestCases. A TestCase is made up of several steps, called TestSteps. The structure of a SoapUI project is like this;

* Project
* Interface
* TestSuites
* TestCases
* TestSteps
* LoadTests
* MockServices

As you can see there is also something called LoadTests, we can gather what they are, but we'll go into detail more what they actually do and how they actually work later.

1. Expand the Sample Simple TestSuite and double click on Simple Login and Logout w. Properties Steps.  
   
2. As you can see the TestCase consist of five TestSteps.  
   
3. You can also click the Description tab in order to see the in test documentation.  
   
4. The step consists of three different types of TestSteps; One PropertyStep, Two TestRequests and Two PropertyTransfer Steps. This is what they do:
   * **PropertySteps:**  
     Stores Properties for later use. In our case it's the Username and Password required to login.
   * **Test Requests:**  
     The actual requests to the server. In this case login and logout.
   * **PropertyTransfers:**  
     A step used for moving properties between different steps. This might be from a Property Step to a request like the step Property Transfer: Move Username and Password does, or to move properties from a Test Request to another, like the step Property Transfer: Move sessionID does.
5. Double click on the TestStep Property Transfer: Move Username and Password.  
   
6. The TestStep Editor will now open on the SoapUI desktop.  
     
   Concept: this property transfer does two transfers in two Properties. First it takes does the TransferUser Transfer. It takes the Property Username from the step Properties: Username and Password and moves it to the step Test Request: login. Then it takes does the TransferPass Transfer. It takes the Property Password from the said step Properties: Username and Password and also moves it to the step Test Request: login.
7. An important feature in SoapUI Tests is The Assertions. Assertions are made in Test Requests and is for validating that the response is what we expected. Open the Test Step Test Request: logout.
8. In the TestStep we have four assertions, asserting different things;
   * **SOAP Response**  
     That the response is a SOAP response.
   * **Schema Compliance**  
     That it complies with the Schema.
   * **Not SOAP Fault**  
     That is isn't a SOAP fault.
   * **XPATH Match**  
     And something called an XPATH match. The XPath Match validates that a certain element has the value expected.

### The Test Run

Now that we have examined the TestCase, let's run it! Press the Play Button and the Test will run and the Test Case progress bar should be full and green.

**Does it turn red after only a step?**  
Don't forget to start the MockService.

**Or... did you run the TestStep earlier?**  
Look at the response, maybe the response is that you already have a session?  
We logged in earlier and aren't allowed to log in again, let's just go to the MockService, stop it and restart it, clearing any server side sessions. Now, rerun the test.

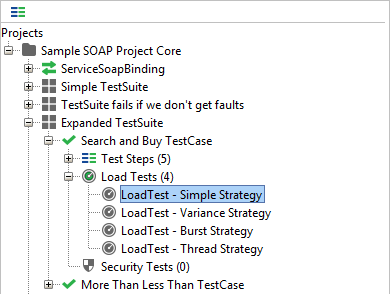
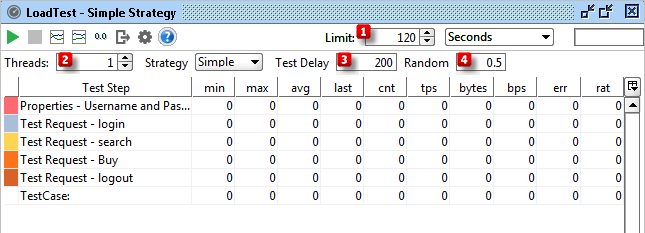
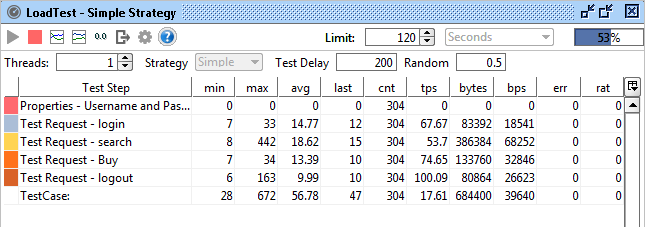
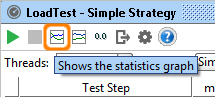
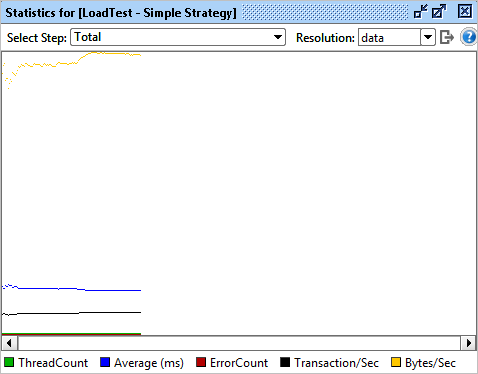
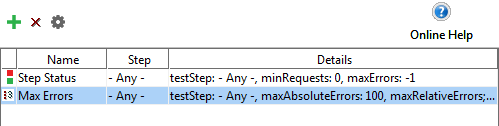
## 4) Web Service Load Tests

### Introduction

Something very useful in SoapUI is the ability to quickly create LoadTests. 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 LoadTest simply by choosing a functional test, right-clicking it and saying New LoadTest. Done!

This makes SoapUI LoadTest extra good since the immediately enable you to load test the functionality not that the Web Service can respond quickly to the same Request over and over again. LoadTests are byproducts of the Functional Tests.

### Tutorial

1. Go to the TestSuite Sample expanded TestSuite and the TestCase Search and Buy TestCase. There are four different LoadTests in that TestCase, one for each LoadTest strategy. For now, we'll choose the LoadTest: Simple Strategy LoadTest. This LoadTest is based on a Load Strategy we call Simple Strategy which is a very basic strategy with a random delay.  
   
2. Let's configure the test.  
   
   1. The first thing to configure in the Simple Strategy is Limit, the Number of second we want the test to run.
   2. The second thing to configure in the Simple Strategy is Threads, the number of threads used. For now we'll use one thread.
   3. The third thing to configure is the Test Delay, which sets the number of milliseconds our base delay will be (1000 is one second). In the test now we have set it to 200 milliseconds.
   4. The fourth item is the Random variable, which sets how the Test Delay should be changed at random, the value 0.5 means it should be half, that is between 100 and 300 milliseconds.  
      What would the value 0 in Random mean? That's right, no random delay at all, just 200 milliseconds all the time.
3. Now, let's run it!
4. As you see the numbers of the test are continuously updated. You can see figures like the response times' throughput per second (tps), assertions, errors, percent of the test run, and more.  
   
5. You can also press the graph button  
     
   and see a graph of the test run.  
     
   Read more about the Load Test Interface here and the different Load Testing strategies here.
6. In the SoapUI LoadTests, you can also do LoadTest Assertions. One of the more important assertions is the Max Errors Step.  
     
   This is used for allowing the maximum number of errors allowed in a test run, which will lead us to the final example, ***discovering an error in our MockService***.

### An error in our MockService

Finally we would like to look at one error we've put in the MockService on purpose.

1. Go to the TestSuite Sample TestSuite fails if we don't get faults and the TestCase TestCase: Searching after Logging out LoadTests. There you will find a load test called LoadTest with Multiple Tests (will fail).
2. Open it and run it. After a time this test will fail