**XML JSON**

**What are APIs?**

APIs (Application Program Interfaces) define how software systems talk to each other, and API documentation is a rapidly growing field. There is a strong need for writers who can understand APIs and explain them so that software developers can understand how to use them. API writers get to be in on the cutting edge of technology in high–paying positions.

**What is JSON?**

**JSON** is a file format that uses human-readable text for storing and transmitting data objects containing attribute-value pairs and arrays. JSON is used to store information in an organized and easy-to-access manner. JSON stands for JavaScript Object Notation. It offers a human-readable collection of data that can be accessed logically.

## What is XML?

**XML** is an extensible markup language that is designed to store data. It is popularly used for transferring data. It is case-sensitive. XML allows you to define markup elements and generate customized markup language. An element is a basic unit in the XML language. The extension of XML file is .xml.

**JSON Code vs XML Code**

Let’s see a sample JSON Code

{

"student": [

{

"id":"01",

"name": "Tom",

"lastname": "Price"

},

{

"id":"02",

"name": "Nick",

"lastname": "Thameson"

}

]

}

Let’s study the same code in XML

<?xml version="1.0" encoding="UTF-8" ?>

<root>

<student>

<id>01</id>

<name>Tom</name>

<lastname>Price</lastname>

</student>

<student>

<id>02</id>

<name>Nick</name>

<lastname>Thameson</lastname>

</student>

</root>

**Advantages of using JSON**

Here are the important benefits/ pros of using JSON:

* Provide support for all browsers
* Easy to read and write
* Straightforward syntax
* You can natively parse in JavaScript using eval() function
* Easy to create and manipulate
* Supported by all major JavaScript frameworks
* Supported by most backend technologies
* JSON is recognized natively by JavaScript
* It allows you to transmit and serialize structured data using a network connection.
* You can use it with modern programming languages.
* JSON is text which can be converted to any object of JavaScript into JSON and send this JSON to the server.

**Advantages of using XML**

Here are significant benefits/cons of using XML:

* Makes documents transportable across systems and applications. With the help of XML, you can exchange data quickly between different platforms.
* XML separates the data from HTML
* XML simplifies platform change process
* Allows creating user-defined tags.

# **Maven pom.xml file**

**POM** is an acronym for **Project Object Model**. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

## Elements of maven pom.xml file

For creating the simple pom.xml file, you need to have following elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| **project** | It is the root element of pom.xml file. |
| **modelVersion** | It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0. |
| **groupId** | It is the sub element of project. It specifies the id for the project group. |
| **artifactId** | It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs. |
| **version** | It is the sub element of project. It specifies the version of the artifact under given group. |

*File: pom.xml*

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
7. **<groupId>**com.javatpoint.application1**</groupId>**
8. **<artifactId>**my-app**</artifactId>**
9. **<version>**1**</version>**
11. **</project>**

Maven pom.xml file with additional elements

Here, we are going to add other elements in pom.xml file such as:

|  |  |
| --- | --- |
| **Element** | **Description** |
| **packaging** | defines packaging type such as jar, war etc. |
| **name** | defines name of the maven project. |
| **url** | defines url of the project. |
| **dependencies** | defines dependencies for this project. |
| **dependency** | defines a dependency. It is used inside dependencies. |
| **scope** | defines scope for this maven project. It can be compile, provided, runtime, test and system. |

*File: pom.xml*

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
8. **<groupId>**com.javatpoint.application1**</groupId>**
9. **<artifactId>**my-application1**</artifactId>**
10. **<version>**1.0**</version>**
11. **<packaging>**jar**</packaging>**
13. **<name>**Maven Quick Start Archetype**</name>**
14. **<url>**http://maven.apache.org**</url>**
16. **<dependencies>**
17. **<dependency>**
18. **<groupId>**junit**</groupId>**
19. **<artifactId>**junit**</artifactId>**
20. **<version>**4.8.2**</version>**
21. **<scope>**test**</scope>**
22. **</dependency>**
23. **</dependencies>**
25. **</project>**

# Data Parameterization Using JSON With Selenium

## ****Introduction****

In testing, it is always important to test application features with different sets of data. Testing with one set of data that we use during recording or while creating a base script will not confirm the functionality of the application alone. For example: When we are doing a login test, it is extremely important to test with all possible valid and invalid credentials to ensure login functionality is working as expected.

One way to achieve testing multiple sets of data is to create individual test scripts/test cases that hold one set of hardcoded data. In this case, if we plan to test a functionality with ten different sets of data, we must create ten independent test scripts with different data. Another way is to create a base script with one set of data and manually change the data and execute the script 10 times.

Both approaches are time-consuming and practically difficult as well as impractical. The best and suggested approach to handling multiple sets of data is to have all the identified test data in any of the data sources like Excel, XML or in JSON files.

Here in this article, we will see how to achieve data parameterization with the help of JSON files.

## ****Case Study****

Mary is a newly-joined automation test developer in your project. She has performed exploratory testing on a Tricentis Demowebshop application and trying to login with multiple users on the application. The test data is provided in JSON format. Can you guide Mary to complete this activity?

### **Expectations**

Help Mary test the login screen with two sets of valid and one set of invalid credentials given in the JSON file.

Report in the JSON file with “Valid user” or “Invalid user” based on the application functionality

**Input JSON File:**

1

[

2

{

3

"users":{

4

"username":"john@abc.com",

5

"password":"abcd@1234",

6

}

7

},

8

{

9

"users":{

10

"username":"doe@abc.com",

11

"password":"pwd@1234",

12

}

13

},

14

{

15

"users":{

16

"username":"henry@abc.com",

17

"password":"abcd@1234",

18

}

19

}

20

]

There are two types of JSON formats, Simple JSON and JSON Arrays.

#### **Simple JSON**

Data are stored here in simple text format which can be accessed easily. The JSON variable we define is an object, which contains many properties using the key:value structure. Refer to this example below for a Simple JSON structure:

1

var mydetails = {

2

"Name" : "Jack",

3

"Age" : "30",

4

"Gender" : "Male"

5

};

#### **Array JSON**

JSON also supports storing multiple sets of data in the form of an Array structure. This can be achieved by keeping multiple objects in square brackets within one JSON structure as below:

1

var empdetails = [{

2

"Name" : "John",

3

"Designation" : "Project Manager",

4

"Gender" : "Male"

5

},

6

{

7

"Name" : "Doe",

8

"Designation" : "Team Lead",

9

"Gender" : "Male"

10

}];

Using an array index of the object empdetails, we can access all elements in the JSON array.

### **How To Work With JSON in Selenium**

There are two ways using which we can load the JSON jar libraries in Eclipse:

1. Download/Add "**json-simple-1.1.jar"** to Java project

2. Use this Maven dependency in pom.xml file of a Maven project

1

<dependency>

2

<groupId>com.googlecode.json-simple</groupId>

3

<artifactId>json-simple</artifactId>

4

<version>1.1.1</version>

5

</dependency>

### **How to Read Data From JSON file**

Create a JSONParserinstance to parse the JSON file into a tree structure with any FileInputStream  object as the parameter

1

JSONParser jsonParser = new JSONParser();

2

FileReader reader = new FileReader("Testdata.json");

3

//Read JSON file

4

Object obj = jsonParser.parse(reader);

 JSONArray is used to parse JSON, which starts with Array brackets

1

JSONArray usersList = (JSONArray) obj;

2

System.out.println("Users List-> "+usersList); //This prints the entire json file

The .get method is used to access the values in the JSON by index

1

for(int i=0;i<usersList.size();i++)

2

{

3

JSONObject users = (JSONObject) usersList.get(i);

4

System.out.println("Users -> "+users);//This prints every block - one json object

5

JSONObject user = (JSONObject) users.get("users");

6

System.out.println("User -> "+user); //This prints each data in the block

7

String username = (String) user.get("username");

8

String password = (String) user.get("password");

9

System.out.println("The username in JSON is: "+username);

10

System.out.println("The password in JSON is: "+password);

11

}

### **How to Write Data to A JSON File**

The "put" method is used to write data to a JSON file.

1

user.put("result", result);

We can use a FileWriterobject and the toJSONString() method to write in a JSON file as a string.

1

file.append(usersList.toJSONString());

## ****Selenium Example to Handle JSON Files****

1

public class JSONHandling {

2

WebDriver driver;

3

​

4

@BeforeTest

5

public void beforeTest() throws IOException {

6

System.setProperty(FirefoxDriver.SystemProperty.BROWSER\_LOGFILE, "null");

7

System.setProperty("webdriver.gecko.driver", "C:\\Selenium\\Drivers\\geckodriver.exe");

8

driver = new FirefoxDriver();

9

driver.get("http://demowebshop.tricentis.com");

10

driver.manage().window().maximize();

11

driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

12

}

13

​

14

@Test

15

public void testAut() throws InterruptedException, IOException, ParseException {

16

readWriteJSON();

17

}

18

​

19

@AfterTest

20

public void afterTest() {

21

driver.close();

22

}

23

​

24

public String login(String username, String password) throws InterruptedException {

25

driver.findElement(By.linkText("Log in")).click();

26

driver.findElement(By.name("Email")).sendKeys(username);

27

driver.findElement(By.name("Password")).sendKeys(password);

28

driver.findElement(By.xpath("//input[@class='button-1 login-button' and @value='Log in']")).click();

29

​

30

if(driver.findElements(By.xpath("//input[@id='vote-poll-1']")).size()>0)

31

{

32

String uname = driver.findElement(By.xpath("//a[@href='/customer/info']")) .getText();

33

if(uname.equals(username))

34

driver.findElement(By.xpath("//a[@href='/logout']")).click();

35

}

36

else

37

{

38

driver.findElement(By.xpath("//a[@href='/login']")).click();

39

return "Invalid User";

40

}

41

return "Valid User";

42

}

43

​

44

​

45

​

46

@SuppressWarnings("unchecked")

47

public void readWriteJSON() throws InterruptedException, IOException, ParseException {

48

JSONParser jsonParser = new JSONParser();

49

try {

50

FileReader reader = new FileReader("Testdata.json");

51

//Read JSON file

52

Object obj = jsonParser.parse(reader);

53

JSONArray usersList = (JSONArray) obj;

54

System.out.println("Users List-> "+usersList); //This prints the entire json file

55

for(int i=0;i<usersList.size();i++) {

56

JSONObject users = (JSONObject) usersList.get(i);

57

System.out.println("Users -> "+users);//This prints every block - one json object

58

JSONObject user = (JSONObject) users.get("users");

59

System.out.println("User -> "+user); //This prints each data in the block

60

String username = (String) user.get("username");

61

String password = (String) user.get("password");

62

String result = login(username,password);

63

user.put("result", result);

64

​

65

//Write JSON file

66

try (FileWriter file = new FileWriter("Testdata1.json")) {

67

​

68

file.append(usersList.toJSONString());

69

file.flush();

70

​

71

​

72

} catch (IOException e) {

73

e.printStackTrace();

74

}

75

​

76

System.out.println(user);

77

​

78

​

79

}

80

​

81

} catch (FileNotFoundException e) {

82

e.printStackTrace();

83

}

84

}

85

}

The output JSON file (Testdata1.json) will look like:

1

[

2

{

3

"users":{

4

"result":"valid user",

5

"username":"john@abc.com",

6

"password":"abcd@1234",

7

}

8

},

9

{

10

"users":{

11

"result":"invalid user",

12

"username":"doe@abc.com",

13

"password":"pwd@1234",

14

}

15

},

16

{

17

"users":{

18

"result":"valid user",

19

"username":"henry@abc.com",

20

"password":"abcd@1234",

21

}

22

}

23

]