**SOAPUI**

**SoapUI:** Used: 5.3.0Current: 5.5.0

**Ready API**: Used: 3.10 Current: 3.2.5

Java –jar Prathap.jar –server.port=8086

**API:** Application Programming Interface

**Web Service:** It is a method of communication b/w two applications or electronic devices over the www. It is collection of operations.

**The difference between API and WebServices is:**

**API:** Pure Methods, which available in the system they given to us like Jars, We don’t have access to their code.

**Web Services:** Over the Network by HTTP call

**Data Collector:** JSON, XML

**Authentication:** Session Key/Token

**i) Soap(simple object access protocol) vs Representational State Transfer (REST)**

|  |  |  |
| --- | --- | --- |
| S.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. |

[**http://216.10.242.178:8082/axis2/services/EmployeeManagementService?wsdl**](http://216.10.242.178:8082/axis2/services/EmployeeManagementService?wsdl)

**SOAPUI**

1. **Assertions for Test validations**

* Contains assertion
* Not Contains assertion
* Soap response assertion
* Http Code-Valid and Invalid
* SLA
* Security

1. **For JSON:**
2. **JSON path Count** : just count of objects
3. **JSON Path Match** : Value assertion
4. **JSON Path Existence Match** – Boolean (True/false)
5. **JSON Path Regex Match**: By giving Regular Expression, we can do true or false
6. **For XML**

**1) Xpath Match:** To check Assertion by using Xpath (To check value EX: age = 26)

1. **To Check particular node is Exists or not (Ex: age):**

* Exists (xpath of node) [Example: **exists(//ns: age)]** andEnter expected result

1. **To Check only one instance of node present:** For example only name node is one

* In above xpath window
* Count(xpath of node) **[example: Count(//ns:name)]** andEnter expected result

**c) To Check Dynamic values:** By using Wildcards give \* inthat particular keep changing value place

**2) X Query Match:** To check whole expression having or not, and we can also check particular value.

* It’s a Boolean, True or False

1. **Features of SoapUI**
2. **GroovyScript TestStep:** We can update xml, json. We can push back into request, trigger test cases.
3. **Script Assertion:** We can only write script for validate your responses**.**
4. **Setup Script:**

* We can use this to execute before everything in that level(Project, Suite, Testcase..)
* Also we can write in groovy using .setSetupScript(“567”)
* EX:testrunner.testcase.testsuite.project.getTSbyname(‘tt’).getTCbyname(‘ff’).setSetupSCript(“g”)

1. **Tear Down Script:**

* Same like above

1. **Load Script**

* Called after the project has been loaded. This can be used to for example initiate some session specific data, endpoints, et c.

1. **Save Script**

Called before the project is being saved, allowing you to perform custom clean-up tasks, for example removing passwords or test items, etc.

1. **Log**
   1. We use this to print something in that particular window.
2. **TestRunner**
   1. We use this in almost every level except Project load and save script level, script assertion window
   2. It will helps to access properties across the project level.
3. **Context**
   1. We use this in almost every level except Project load and save script level
   2. Used to access and modify the properties of test cases, it will work inside the particular test case only.
4. **Message exchange** 
   1. It stores all the details of last request and response
   2. We use this in Script Assertion window
      1. def response = **messageExchange.response.responseContent**
      2. def responseJSON = new JsonSlurper().parseText(response);
   3. **Some Assertion Examples:**
      1. **assert(messageExchange.**timeTaken>500)
      2. **log messageExchange.**getEndPoint
      3. **assert (messageExchange.responseHeaders[“Content-Length”]!= null);**
      4. **Assert (messageExchange.responseAttachments.length ==0;** (If we have we can pass no.of attachments.)
5. **Parameter Types:**
   1. **Path Parameter:** Which comes along with base url, it’s kind of table in DB
   2. **Query Parameter:** Which comes after? mark, it’s kind of where condition in DB
   3. **Template:** It’s also like Path parameter, variable parts of resource path.
   4. **Header:** To pass custom headers data through Request
   5. **Matrix:** These parameters added end of the resource path and separated by semi colon, these don’t make any change in response of service
   6. **Plain:** To omit the parameters without delete them from resource path, when we want we can activate.
6. **Attachments:**

* Content Type should match with request which server accepts
  1. **Multi-Part Upload**
     1. In this request (file) is divided into multiple files/data sets and send to server
     2. Content Type of this messages :
        1. Multipart/form-data: If application uses HTML form data
        2. Multipart/mix: Except HTML
  2. **Simple Upload:**
     1. Sending a file in request body
     2. Content Type of request is indicate file type
  3. **Cache Attachment:**
     1. It save the attachment in Project.xml
     2. When we send this project to someone or version control, no need to send attachment separately
     3. It increase little memory of project
  4. **Content ID:**
     1. If attachment is related to a Query Parameter
     2. Give that query parameter in Content ID, so these can integrate
  5. **Avoid Attachment:**
     1. We can also avoid attachment of file by giving file path in query parameter
        1. EX: file${projectDir}\overlay.jpg
     2. File should be at project level, so if we send(with file) to others also it works

1. **Authentication & Authorization:**
   1. **Authentication**
      1. When client interacts with server with protected resource, client should prove its identity, it’s authentication
      2. In headers we can see, what kind of authentication service accepts
   2. **Authorization**
      1. To check client permitted to perform an particular action
         1. EX: Cline can insert data but cannot delete (read only, no write)
      2. Authorization takes place after Authentication
2. **Data-Driven Testing:**
   1. Using Excel files
   2. And other forms allowed in Pro (like data generator, JSON, XML…)
3. **SoapUI Pro Features:**
4. **Data source**

Uses for store data manually or store from excel sheet

1. **Data source loop**

Uses to run multiple sets of data which we stored/imported from excel in data source step

1. **Data Sink**

Which we use to export response to an excel sheet

1. **Data Gen**

To randomize input to parameter use, it uses any value from list

* 1. List
  2. Number (Random numbers, with interval 2 etc..)
  3. Template 🡪 It does concatenate the lines data

1. **Test Case Coverage**

Depends on WSDL or Rest Resources, how many services we utilized from them in %.

1. **Test Case Debugging/Break Point**

By keeping a break point at test step in test case we can debug the issue. We get star if parameter changes in step.

1. **Test on Demand**

We can run our test cases in different locations (UK, Chicago etc..)

1. **Test Case Requirement**

Giving US#, AC etc.. We can store related to test case in future we don’t get confuse

1. **Create Report:**

Have 3 types of reports

* 1. Test case report
  2. JUnitStyle-HTML report
  3. Data export

1. **Record and play**
2. **Transaction logs, History, Compare, Project Coverage at Log level of Project**
3. **Environment Setup**
4. **Auth Manager:** Add custom Authentication, which we can use for other requests also, which having same, like parameterization
5. **We can also create Test cases for all operations of a Definition (EX:Swagger or WSDL) including basic assertions at a time for all.**
6. **In JSON assertions, It’s creates JSON path automatically, when we select desired element**
7. **Properties in SOAPUI:**

**1. Project/Test Suite/ Test Case Level Properties Access:**

Syntax for Test suite: ${#Testsuite#property} (ex: ${#testsuite#name})

Syntax for Test Case: ${#Testcase#property} (ex: ${#testcase#id})

Syntax for Project: ${#Project#property} (ex: ${#Project#id})

**2. Properties Level:**

If you would like to give all data(properties) in single sheet then we have to use This properties level

To retrieve this go to particular request and right click

* Get data
* Step2[properties]
* Property[name]

Syntax: ${Properties#Department}

**This is not generic keyword from soapui, so we don’t have # like above**.

**3. Properties upload from External file:**

* Browse file & Add file & And use

**4. Property Transfer Functionality:**

When we would like to clone all test cases into single step, we use this functionality.

**For JSON:**

* We can get element from JSON response (use pathfinder online)
* And pass the path in source
  + Ex: $[0].slicAbbreviation
  + Ex2: we can write multiple ways
    - location[2].slicAbbreviation
    - .location[2].slicAbbreviation
    - $.location[2].slicAbbreviation
* Also select Target path

**For XML:**

* And go to your source (get employee response window)
* Copy ns url(ex: ns=<http://sample.com/reservation/guest/types>)
* Go to property transfer window
* Click on ns on top two green signals(>>)
* Then enter ns url in source
* And write Xpath for name property(which property u are going to use)
* EX: **//ns:getEmployeeDetailsResponse/ns:return/ns:name/text()**
* {text() is for grab the name(reddy) from xpath)
* And give above url in Target window also
* Select property as request(because we have to send above name in delete request)
* And give xpath where above text() has to go

**We can achieve property transfer by manually:**

If I would like to name from add employee to get employee.

We can write like

${fromtestcasename #whereyouare getting text #xpath of text}

**EX: Syntax**

**${addemployee#Request#//typ:addEmployee/typ:name/text()}**

1. **Database Testing:**

Crete a data base.

CREATE DATABASE DATABASENAME

EX**: CREATE DATABASE** PRATHPDB

USE DATABASENAME

CRAETE TABLE Training (name VarChar(20), course VARCHAR(20), location VARCHAR(20), skill VARCHAR(20));

INSERT INTO Training VALUES(‘prathap’, ‘SopaUI’,’US’, ‘Perf’);

1. **JDBC In SoapUI:**
2. **Driver:**

**MySQL**: com.mysql.jdbc.Driver

**MSSQL**: com.microsoft.sqlserver.jdbc.SQLServerDriver

1. **ConnectionString :**

**MySQL:** jdbc:mysql://localhost:3306/employeemanagment?user=root&password=root

**MSSQL:**jdbc:sqlserver://Csvt0000ca35.tus.ams1907.com:23100;user=tmeconnect;password=TmEdatabase2018

1. **Assertion:**

**JDBC status**: JDBC executed properly or not

**JDBC Timeout**: Is query executed in specified time or not. We can specify time at test step custom properties

* **We can also parameterize sql query depends on custom properties using “GetData” like ${#TestCase#Name}**

1. **SoapUIPro feature for DB:**

* **Build Query**: If we select rows, soapui builds query for us
* Every JDBC Step as a option “Response as xml”, which gives response in XML.
* By using this we can pass a value from other test steps to one test step

In query, by giving ‘’ and right click in b/w will select response as xml and select value

1. **Performance Testing:**

* SoapUI suggested use with 100 concurrent users. Better to use Load UI(License)
* Have to select strategy what kind of test we need.
* **Simple**: Specific number of threads, without fluctuations.
* **Burst**: Burst of time. Lot of users hit at time, ramp up
* **Threads**: Amount of threads, with increment, aggregate depends on time.
* **Variance**: Randomize amount of load

1. **Threads**: Amount of concurrent users hit services at same time
2. **Test Delay:** wait time b/w concurrent users.

**Ex:** If we give 20 users, after execute 20, it wait amount of breathing time, then it execute again.

1. **Random:** The random factor of the Test Load. With a Test Delay of 600 ms and a Random factor of 0.5, the actual delay will be uniformly distributed between 300 ms and 600 ms. The actual delay can thus be calculated as *TestDelay - RandomNumberBetween( 0, TestDelay\*Random )*.
2. **Limit:** time for load on services

**Have different validation for load test**:

Step TPS

Max Errors

Step Avg:

Step Max

Step Status

1. **Jenkins Integration:**

* Click on Launch Test runner on rig click at Project/Testsuite..
* Copy the below commands from test runner window
  + **directory**: C:\Program Files (x86)\SmartBear\SoapUI-5.5.0\bin\
  + command: cmd.exe /C testrunner.bat G:\UPS\SoapUIGIT\tam\_asset\_management\_service\TransportationAssetManagementSystem-soapui-project.xml
* Get Jenkins from site, down load Jenkins.war file
* Goto CMD and Navigate to war file path
* Run that war file on a port (**java –jar Jenkins.war –httpPort=9090**)
* Go to browser and **http:// localhost:9090**
* Add all suggested plugins
* Create a project with **“Free Style”** Project
* Go to “**Build Section**” of project and add all details like post build etc..
* Select “**Execute Windows Batch Command**” from drop down
* Give The above command which we copied from Test Runner path
  + C: (Give C: just to make sure, if cmd somewhere it automatically comes here before it executes below command)
  + Cd C:\Program Files (x86)\SmartBear\SoapUI-5.5.0\bin\testrunner.bat G:\UPS\SoapUIGIT\tam\_asset\_management\_service\TransportationAssetManagementSystem-soapui-project.xml
* Build Job Now

1. **API Monitoring**

* Checking of API is at regular interval for
  + Availability
  + Correctness
  + Performance
* Goto Project/RighClick/ MoniterAPIs
* We get **AlertSite** window then sigin/signup
* After connect click on “**Add Monitor**”
* Then add details like Project, Test Suite, Test Case etc..
* After Done, click on “**API Status**” at Top Right corner
* We can click on link, it take us to “Alert site website”, we can see all details there like errors, issues..
* Have Great DashBoard like Openshift, Splunk in “AlertSite” with all details /data

1. **Groovy Basics:**

**To retrieve the data by using testrunner from different test case level, we have to write like:**

TestRunner.testCase.testSuite.testCases["whichtestcaseyouliketogetdata(Ex:getEmp)"].getPropertyValue("id")

**If you have property value in same test case then:**

testRunner.testCase.getPropertyValue(“id”)

**If we have to get property from test step of other test case:**

testRunner.testCase.testSuite.testCases["GetEmp"].testSteps["Get"].getPropertyValue("Request")

**To set value in groovy by using Set:**

testRunner.testCase.testSuite.testCases["GetEmp"].setPropertyValue("id", 4567).

**To Retrieve Custom Property from project level:**

def Project = testRunner.testCase.testSuite.project

Project.getPropertyValue("Gender")

1. **Loops:**
   * **For:**

For(def i=0; i<5; i++){

Log.info I;

}

* + **While**

def j = 1;

While(j<5){

J++

}

1. **Array**
   * To store multiple values with Index

Def terms = [“SoapUI”, “ReadyAPI”, “Groovy”]

Log.info terms[0] //soapui

For(def i=0; i<terms.size(); i++){

Log.info terms[i]

}

1. **List**
   * It’s for set of values

Def courses = [1,2, “Soap”, 3,4]

Log.info course[2] //soap

1. **Array List**
   * It’s same like array, but no index required to store value

ArrayList ar = new ArrayList()

Ar.add(“SoapUI”)

Ar.add(“ReadyAPI”)

Log.ifo ar.get(0) //SoapUI

1. **Hash Table**
   * To store key value pair
   * To get a particular value, we use key value to get that
   * Hash Table does not allow null keys or values.

HashTable ht = new HashTable()

Ht.put(“Name”, “SoapUI”);

Ht.put(“ID”, “5.5.0”);

Log.info ht.get(“Name”) //SoapUI

* + Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.

1. **Hash Map**
   * It’s also like Hash table, to store key value pair
   * But no need to use Put or get commands to store and retrieve values from list
   * Hash Map allows one null key and any number of null value.

Def names=[“Name”:”SoapUI”, “ID”:”5.5.0”]

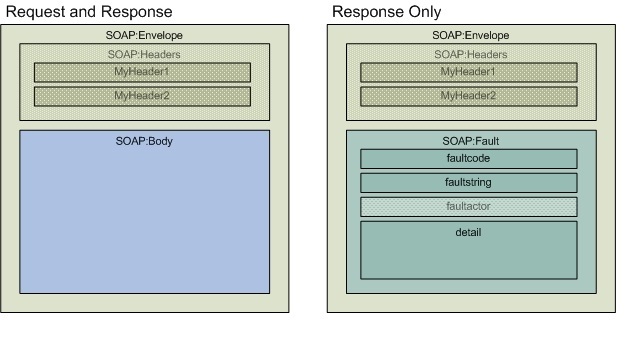
Log.info names[“Name”] //SoapUI

1. **Try Catch**
   * It’s to handle exceptions
   * Same like Java, there is no difference
2. **SOAP**

It is a protocol for sending and receiving messages b/w applications without confronting interoperability issue.

**WSDL(Webservices Defination Language):**

* It is an contract for web services
* It has location of web services
* It has different operations what that particular web service support.
* It has data types of using in web services, data elements for each elements of that.
* It is in XML format in below structure.



**Types of web service:**

**Structure Of Soap:**

Soap Envelope

Soap Header

Soap Body with call and response data.

Soap Fault with errors and status data.

**REST Manual In SOAPUI**

**HTTP Methods:**

C—Create--POST  
R—Retrieve--GET  
U—Update--PUT  
D—Delete—Delete

HTTP Request:

* URI
* Headers
* JSON/XML Payload

HTTP Response:

* Status-200 ok
* Response Payload
* String Message-successful

**POST: To Create new Entity in server or DB**

* JSON/XML form

**GET: To Retrieve (Fetch) Data from server/DB**

* URI
* Path Parameter
* Query Parameter
* Headers

**PUT: To Update the info**

**DELETE: To delete data.**

**URI (Uniform Resource Identifier) = URL+HOST+PORT+(Path parameter/query parameter)**

**URI= URL(Endpoint URL)+ (API) Service URL**

**EX: https://reqres.in/api/users?page=2**

**? = Query parameter**

**Page : Parameter**

**=2: parameter Value**

**Status Response Code:**

* **200 Ok**
* **201 Created success**
* **400 Bad Request**
* **401 Unauthorized(authentication error)**
* **404 Not Found**
* **500 internal Error**

**JMETER:**

**TestPlan** : select Thread Group give details like no.of threads, **ramup period**, loop count.

Then go to **sampler** and HTTP Request.

And give details like server name, port, path(with XML/JSON) and parameters.

Then add **Listener** with Result Tree/aggregate report/Graph results

**WorkBench**

**SOA VS Web Services:**

1. **Monolithic (Single Unit):** Hosting number of components, hosted together and deliver together. It’s not scalable. It’s time taking and not fit for complex architecture, we cannot use diff technologies at a time.
2. **SOA (Coarse-Grained Architecture):** The features brokendown into smaller components, each feature may contain number of services.
3. **Micro Services (Fine-Grained):** The services broken down into task level pieces**.**

**EX:**

**SOA** is like an orchestra where each artist is performing with his/her instrument while the music director guides them all.

**Micro Services**: Dance group, each dancer independent and they know what they need to do. If they miss some steps they know how to get back.

**Difference b/w with respect Certain Parameters:**

1. **Architecture and Co-ordination Difference:**

**SOA Have:**

1. **Business Service:** Its core business operations, represented by XML/WSDL.
2. **Enterprise Service:** Implements the functionality defined by Business service**,**
3. **Application Service:** This is core functionality of the feature, can be invoked directly or through UI.
4. **Infrastructure Service :** It goes to non-technical/Non-functional features, like security, performance and scaling

These 4 services together form SOA/deliver a software product

**Micro Services Have:**

1. **Functional Service:** It’s a combination of Business, enterprise and application services.
2. **Infrastructure Service:** It goes to non-technical/Non-functional features, like security, performance and scaling.
3. **Heterogeneous Interoperability:**

**SOA:**

If Services are different technologies, it’s difficult to interact to one and other, so we need Messaging middle ware. It act as bridge b/w different apps which are in different languages.

**Micro Services**:

No need any broker, it can communicate any kind of apps with API Layer.

1. **Service Granularity:**

**SOA:**

The service consumer doing one operation, multiple services call at one time

**Micro Services:** One operation may involves multiple services or called by one and other. **1>2>3**

1. **Component Sharing:**

**SOA:** We have a service which pull data from different DBs and give to user for different features.

**Micro:** We have different services which pull data from different DBs and give to user for different features.