**Introduction :**

***Introduction to SoapUI:***

1. Leading web services testing tool.
2. It is open source and free to use.
3. Companies expose their functionality via wen services. Mobiles, laptops exchange data from different services via web services.

***Introduction to Web Services:***

1. A piece of software that can be called over internet.
2. Two types of web services :
3. SOAP services : XML(communicate through)
4. RESTFul services : XML or JSON(communicate through)
5. Service provider : The one who provides service.
6. Service consumer : The one who consumes service.

***Introduction to XML and JSON:***

1. XML / JSON marks meaningful data for interpretation.
2. XML / JSON wraps data with metadata for making meaningful data.

***Introduction to WSDL:***

1. Web Service Description Language
2. It is technical document which is used by the web service developers/testers.
3. Developers refer for developing web services. Testers use for SoapUI testcase generations.
4. It is xml document which contains some technical information about web services.
5. It contains the available operations. Input/output of each operation. Datatypes of input and output variables. Address of the web service.
6. Address – SEI (Service Endpoint Interface)
7. Sample WSDL file : <http://www.dneonline.com/calculator.asmx?wsdl>

***Introduction to SOAP:***

1. Simple Object Access Protocol
2. SOAP is used in web service communication.



1. Root element must contain ‘envelope’ tag
2. Header field consists of some technical information. And it is not mandatory.
3. Body part is mandatory. It contains actual payload. In the above example payload is ‘authenticate’ operation.

***Mock Services:***

1. Mock service we need to start (right click and start minimized to run mock service).
2. Mock service will run on local machine after start.
3. It works as a normal web service.
4. Mock service mimics behaviour of the actual web service.
5. Double click on mock service and below section(MockService properties) we get properties like Description, Path, Port etc.
6. SEI we need to change to run mock service.

Ex :- <http://localhost:8080/BLZLocalService>

1. Right click and select ‘stop’ to stop mock service.

***XPath Expressions:***

1. XPath means XML path.
2. Used to extract contents from XML document.
3. XPath expressions are helpful while writing assertions for web service response.
4. Ex:- //Employee/Address/HomeAddress/StreetName
5. Ex: - //ns1:GetSupplierByZipCodeResponse/ns1:SupplierDataLists/ns1:SupplierDatas/ns1:SupplierData
6. Ex :- count(//ns1:GetSupplierByZipCodeResponse/ns1:SupplierDataLists/ns1:SupplierDatas/ns1:SupplierData[ns1:ZipPlus4!=’75070’])

**Assertions :**

***Assertions Introduction:***

1. Assertions are needed to validate content of responses.
2. Compliance assertions to make sure that web service responses are as per standards.
3. This also includes XPath expressions.
4. By default, compliance SOAP assertion is added in the SoapUI tool.
5. There are property content; compliance, status and standard; Script; SLA; JDBC; security assertions.
6. **Property content :** contains, not contains, XPath match, XQuery match, JsonPath Count, JsonPath Match
7. **Compliance, status and standards :** Invalid http status codes, valid http status codes, not soap fault, Schema compliance, WS-security status, SOAP fault
8. **Script :** Script assertion
9. **SLA :** Response SLA
10. **Security :** Sensitive information exposure.

***XQuery Assertion:***

<zipcodes>

{

For $supplier in //ns1:SupplierData

return <zip>{data($supplier/ns1:ZipPlus4/text())}</zip>

}

</zipcodes>

***Handling CData:***

1. Any xml message / bytes of info / raw data, within xml are put in CData.
2. So that web service will differentiate normal data and CData.

***Groovy for handling CData:***

Def response = context.expand(‘${getAirportInfo#Response’) -> prints complete response of step ‘getAirportInfo’.

Log.info response

***XmlSlurper:***

1. When we extract response of any step using groovy, it is in the form of text instead of xml.
2. To convert normal text to xml we need XmlSlurper class.

Import groovy.utils.XmlSlurper

XmlSlurper xmlSlurper = new XmlSlurper()

def xmlData = xmlSlurper.parseText(response)

log.info xmlData.Table[0].AirportCode

String airportcode = xmlData.Table[0].AirportCode

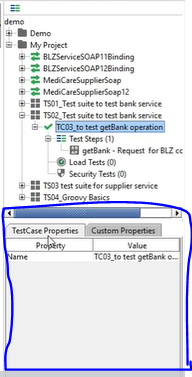
Assert airportcode == ‘LHR’

Log.info ‘Assertion Done’

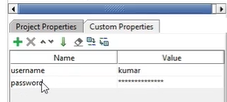
**Properties :**

***Introduction to properties:***

1. Property is a placeholder to store data at runtime during execution of test cases.
2. Properties are like variables in programming languages.
3. Properties are at project level, test suite level and test case level.



1. TestSuite Properties / TestCase Properties are implicit properties created by SoapUI. We cannot create properties under these tabs.
2. If we want to create our own properties we need create under ‘custom properties’.
3. **Implicit properties(test case level) :** Name, description, message size, encoding, endpoint, timeout, bind address, interface, operation, username, password, domain, authentication type etc.
4. **Create project level custom properties :**



1. Select Project level custom properties -> add ‘username’ in the Name and its value.
2. Repeat same for password.
3. If we create properties at project level, all test suites and test cases can access.
4. If we create properties at test suite level, only test cases in that suite can access.
5. We create properties at project level which we think used by all tests. Which reduces duplicate work.
6. Project level property we access like this, **${#Project#username}**

***Properties Expansion:***

1. General syntax,

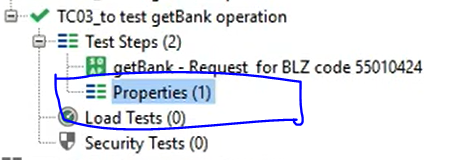
**${ <scope> <property name>}**

1. Scope can be one of the following literal :

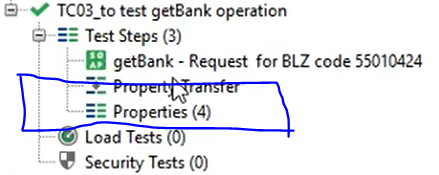
#Project#, #TestSuite#, #TestCase#, #Global#

***Properties Transfer:***

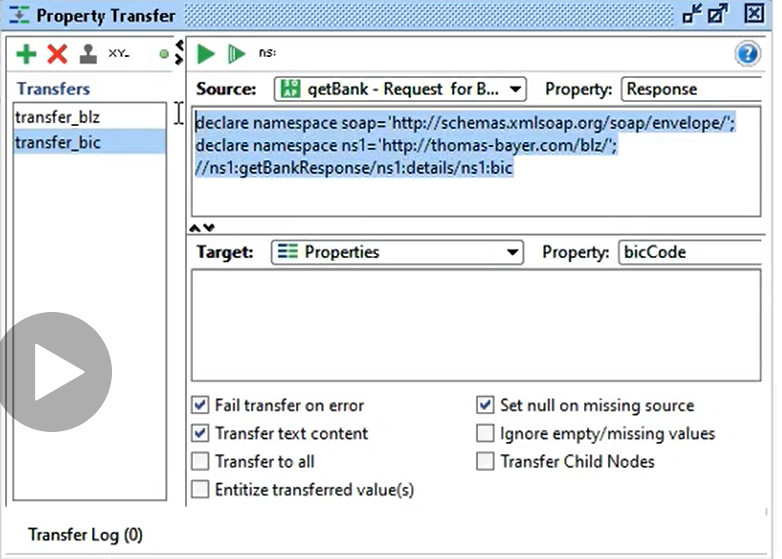
1. We can add ‘properties’ step,



1. It is used for storing property values from previous response for future use.



1. Response from test step -> ‘PropertyTransfer’ step transfers properties -> Properties are saved in ‘Properties’ step
2. ‘PropertyTransfer’ step has 3 sections; transfer name, source of transfer and destination of transfer.



**REST Services :**

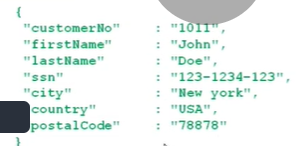
***Introduction to REST Services:***

1. Representational State Transfer.
2. Web standards based architecture and uses HTTP protocol for data communication.
3. In REST architecture, a REST Server simply provides access to resources using URIs through browser URLs.
4. Communicate using Text, JSON and XML. JSON is the most popular format being used.

Sample request :



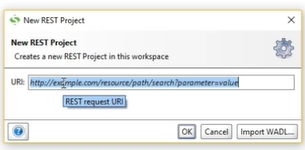
Sample response :



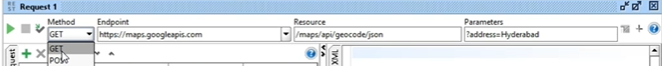
1. There are no protocols like SOAP, we can call easily with normal URL and get plain JSON/XML response.

***Creating REST Project:***

1. FILE -> New REST Project



1. Which prompts us to use REST URI.
2. Project created
3. REST interface is created.
4. REST supports GET, POST, PUT and DELETE methods



1. URI is split into 3 parameters Endpoint, Resource and Parameters
2. By default the assertion part is empty. Where as for SOAP request by default there is SOAP compliance assertion.
3. Sample JSONPath expression,

$.status ($ means root. Here after root status element)

1. For JSONPath expression we can use third party tools or browser plugins for lengthy JSON contents.
2. SoapUI-pro has automatic JSONPath expression creation.
3. JSONView plugin for chrome for the same.
4. Go to the plugin and paste JSON. Right click on the element for which we want JSONPath and copy it. Paste in the SoapUI tool along with “$.”

***REST Methods:***

1. **GET -> Read data from the REST resource**



(here we are sending as query parameter)

1. **POST -> Create data**



(here we are sending whole json data)

1. **PUT -> Update data**



1. **DELETE -> Delete data**

