7. Parsing XML & schema Validations

JSON vs XML in API Development

- ➔ Nowadays, it is very rare to see XML being used. Most companies and developers prefer to develop APIs using JSON instead of XML because of below reasons
  - ◆ JSON is much faster for data transfer between client and server.
  - ◆ JSON is lighter and simpler compared to XML.
  - ◆ Encryption and decryption are easier with JSON format.
  - ◆ XML is heavier and more complex to handle.
- ➔ **Tools used**
  - ◆ JSON → JSONPath
  - ◆ XML → XMLPath

What is XML?

- ➔ XML stands for Extensible Markup Language and is case-sensitive.
- ➔ Commonly used for configuration files, data exchange, and web services (SOAP APIs).
- ➔ It is used to store and transport data and both human-readable and machine-readable.
- ➔ Uses tags like HTML but is designed to describe data, not display it.
- ➔ We can define our own tags — it is extensible.

XML Rules

- ➔ Every opening tag must have a closing tag.
- ➔ Tags must be properly nested.
- ➔ XML document must have one root element.
- ➔ Attributes must be quoted.

Simple XML Example

```
<employee>
    <id>101</id>
    <name>John Doe</name>
    <department>IT</department>
</employee>
```

- ➔ **<employee>** → Root Element (or Parent Element) or Start Tag
- ➔ **<id>** → Child Element of <employee>
- ➔ **id** → Element Name
- ➔ **101** → Element Value (or Text Content)
- ➔ **<//employee>** → End Tag
- ➔ **Together: <id>101</id>** → XML Element

XML Example with attribute

```
<user id="1" role="admin">Madhan</user>
```

- ➔ **<user>** → Root Element (or Parent Element) or Start Tag
- ➔ **id="1"** → Attribute (id is the name, 1 is the value)
- ➔ **role="admin"** → Attribute (role is the name, admin is the value)
- ➔ **Madhan** → Element Value (or Text Content)
- ➔ **</user>** → End Tag

➔ **Together: <user id="1" role="admin">Madhan</user>** → XML Element with Attributes and Value

Workout in Postman Tool

➔ Import **JSON And XML Schemas and Responses.json** collection.

➔ Observe XML responses of API's

➔ Postman does not directly support XML Schema (XSD) validation directly. We need to first convert the XML response to JSON using **xml2Json()**, and then perform validations using standard JSON assertions.

```
<user>
    <id>101</id>
    <name>John</name>
</user>
```

```
const responseBody = pm.response.text();
let xmlData = xml2Json(responseBody);  // Converts XML to JSON
pm.test("Check user ID", function () {
   pm.expect(xmlData.user.id).to.eql("101");
});
```

test_parsing_xml.py

```
import json
import requests
import xmltodict
from xml.dom.minidom import parseString
class TestXMLParsing:
```

Test 1: Basic XML Element Validation

```
  @pytest.mark.run(order=1)
  def test_xml_response_1(self):
     """
     Validates:
     - HTTP status code
     - Content type
     - Specific XML element values
     """
     url = "https://mocktarget.apigee.net/xml"
     response = requests.get(url)
```

Status code & content-type validation

```
     assert response.status_code == 200, "Status code should be 200"
     assert response.headers["Content-Type"] == "application/xml; charset=utf-8", "Content-Type mismatch"
```

Pretty print raw XML

```
     print("------ Pretty Printed XML ------")
     print(parseString(response.text).toprettyxml())
```

Convert XML to dictionary

```
    json_data = xmltodict.parse(response.text)
                                        Pretty print XML as JSON-like format
    print("------ XML Parsed to JSON Format ------")
    print(json.dumps(json_data, indent=4))
                                        Extract and validate values
    root = json_data["root"]
    assert root["city"] == "San Jose", "City should be San Jose"
    assert root["firstName"] == "John", "First name should be John"
    assert root["lastName"] == "Doe", "Last name should be Doe"
    assert root["state"] == "CA", "State should be CA"
                                        Test 2: XML Attribute Validation
  @pytest.mark.run(order=2)
  def test_xml_response_2(self):
    """
    Validates:
    - HTTP status code
    - Content type
    - XML attributes using '@' notation
    """
    url = "https://httpbin.org/xml"
    response = requests.get(url)
                                        Status code & content-type validation
    assert response.status_code == 200, "Status code should be 200"
    assert "application/xml" in response.headers["Content-Type"], "Content-Type should be
application/xml"
                                        Pretty print raw XML
    print("------ Pretty Printed XML ------")
    print(parseString(response.text).toprettyxml())
    pretty_xml = parseString(response.text).toprettyxml()
    cleaned_xml = "\n".join([line for line in pretty_xml.split('\n') if line.strip()])
    print("------ Cleaned Pretty XML ------")
    print(cleaned_xml)
                                        Convert XML to dictionary
    json_data = xmltodict.parse(response.text)
                                        Pretty print XML as JSON-like format
    print("------ XML Parsed to JSON Format ------")
    print(json.dumps(json_data, indent=4))
                                        Extract and validate attributes
    slideshow = json_data["slideshow"]
    assert slideshow["@title"] == "Sample Slide Show", "Title should be 'Sample Slide Show'"
    assert slideshow["@date"] == "Date of publication", "Date should be 'Date of publication'"
```

```
    assert slideshow["@author"] == "Yours Truly", "Author should be 'Yours Truly'"
                                    Test 3: Parsing and Validating Slide Content
  @pytest.mark.run(order=3)
  def test_parsing_xml_response(self):
    """
    Validates:
    - Number of slides
    - Slide titles
    - Number and content of items
    - Dynamic presence check
    """
    url = "https://httpbin.org/xml"
    response = requests.get(url)
                                    Status code & content-type validation
    assert response.status_code == 200, "Status code should be 200"
    assert "application/xml" in response.headers["Content-Type"],  "Content-Type should be
application/xml"
                                    Pretty print raw XML
    print("------ Pretty Printed XML ------")
    print(parseString(response.text).toprettyxml())
    pretty_xml = parseString(response.text).toprettyxml()
    cleaned_xml = "\n".join([line for line in pretty_xml.split('\n') if line.strip()])
    print("------ Cleaned Pretty XML ------")
    print(cleaned_xml)
                                    Convert XML to dictionary
    json_data = xmltodict.parse(response.text)
                                    Pretty print XML as JSON-like format
    print("------ XML Parsed to JSON Format ------")
    print(json.dumps(json_data, indent=4))
                                    Extract and normalize slides
    slides = json_data["slideshow"]["slide"]
                                    Validate slide titles
    titles = [slide["title"] for slide in slides]
    assert len(titles) == 2, "There should be 2 slide titles"
    assert titles[0] == "Wake up to WonderWidgets!", "Wrong First title"
    assert titles[1] == "Overview", "Second title should be 'Overview'"
    assert "Overview" in titles, "'Overview' should be in titles"
                                    Validate items
    items = []
    for slide in slides:
      item = slide.get("item", [])
```

```
        if isinstance(item, str):
            items.append(item)
        else:
            items.extend(item)
    assert len(items) == 3, "There should be 3 items"
    assert items[0]['em'] == "WonderWidgets", "First item should be 'WonderWidgets'"
    assert items[2]['em'] == "buys", "Last item should be 'buys'"
```

Check for presence of specific item dynamically

```
    found = any("WonderWidgets" in item.values() for item in items if isinstance(item, dict))
    assert found, "'WonderWidgets' should be present in items"
```

Note

- ➔ Use **xmltodict** when you want to easily convert XML to a dictionary and validate values just like working with JSON.
- ➔ Use **lxml** when you need to use XPath, parse complex XML, or handle attributes, namespaces, or large XML files.

XML Schema Validation

- ➔ A schema is like a blueprint or template for data.It tells
  - ◆ What fields are expected
  - ◆ What data types they should be
  - ◆ Which fields are required or optional
- ➔ It is used to validate if JSON or XML data is correct.
- ➔ **XML Schema (XSD)** is used to define the structure of XML.
- ➔ **JSON Schema** is used to define the structure of JSON.
- ➔ In **JSON And XML Schemas and Responses.json** collection observe Validations on **Post Response script** tab.

Online Tools for Schema Convertion

- ➔ **JSON to JSON Schema converter**
  - ◆ Free Online JSON to JSON Schema Converter
- ➔ **XML to XSD converter**
  - ◆ https://www.freeformatter.com/xsd-generator.html#before-output
  - ◆ https://www.site24x7.com/tools/xml-to-xsd.html
- ➔ **Convert XSD to JSON Schema**
  - ◆ https://www.convertsimple.com/convert-xsd-to-json-schema/

test_schema_validations.py

```
import requests, json
import xmlschema
from jsonschema import validate, ValidationError
class TestSchemaValidation:
```

Test 1: JSON Schema Validation

```
    def test_json_schema_validation(self):
        """
```

```
    Validate JSON response against schema
    """
    url = "https://mocktarget.apigee.net/json"
    response = requests.get(url)
    assert response.status_code == 200, "Status code should be 200"
                                        Load response and schema
    data = response.json()
    with open("./jsonSchema.json", "r") as f:
        schema = json.load(f)
    try:
        validate(instance=data, schema=schema)
        print("JSON schema validation passed")
    except ValidationError as e:
        print("JSON schema validation failed:", e)
        assert False
                                    Test 2: XML Schema (XSD) Validation
def test_xml_schema_validation(self):
    """
    Validate XML response against XSD schema
    """
    url = "https://mocktarget.apigee.net/xml"
    response = requests.get(url)
    assert response.status_code == 200, "Status code should be 200"
                                            Load XML schema
    schema = xmlschema.XMLSchema("./xmlSchema.xsd")
    try:
        schema.validate(response.text)
        print("XML schema validation passed")
    except xmlschema.validators.exceptions.XMLSchemaValidationError as e:
        print("XML schema validation failed:", e)
        assert False
```