**Architecture of Http-JSON Integration:**

The Http-JSON is implemented using the third party library “HttpClient”.

HttpClient is used to provide the core HTTP functionality. Some of the features provided by the HttpClient are

Some of the features of HTTP Client are:

1. Full implementation of all HTTP methods (GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE) in an extensible OO framework.
2. Supports encryption with HTTPS (HTTP over SSL) protocol.
3. The ability to set connection timeouts.
4. Http Methods implement the Command Pattern to allow for parallel requests and efficient re-use of connections.
5. Automatic Cookie handling for reading Set-Cookie: headers from the server and sending them back out in a Cookie: header when appropriate
6. Direct access to the response code and headers sent by the server.

The general process for using *HttpClient* consists of a number of steps:

1. Create an instance of HttpClient.
2. Build URI using the parameters that are passed and depending on the method type choose the appropriate method to execute.
3. Tell HttpClient to execute the method.
4. Read the response.
5. Release the connection.
6. Deal with the response.

**Instantiating HttpClient**

The no argument constructor for HttpClient provides a good set of defaults for most situations so that is what we'll use.

HttpClient client = new HttpClient();

**Creating a method Instance based on Method type**

Depending on the method type the appropriate execute method is called.

**Build URI**

To create a valid URI the following parameters are required.

1. Scheme = Scheme name ("http" specifies the 'scheme' name)
2. Host = Domain name of the Application.
3. Port = Port number
4. URL= URL is the application URL that can be used to perform any functionality.
5. Query= Query String would be the parameters that are required by the application to complete a particular request.
6. Fragment = Fragment (usually this is null)

URI uri = URIUtils.createURI(scheme, host, 80,url , query, null);

## Execute the Method

The actual execution of the method is performed by calling execute method on the client and passing in the method to execute.

The other useful piece of information is the status code that is returned by the server. This code is returned by execute method as an int and can be used to determine if the request was successful or not and can sometimes indicate that further action is required by the client such as providing authentication credentials.

The following execute methods have been implemented in our integration.

1. Get –

HttpGet httpget = new HttpGet(uri);

response = httpClient.execute(httpget, localContext);

1. POST –

HttpPost httppost = new HttpPost(uri);

response = httpClient.execute(httppost, localContext);

**Read the Response**

It is vital that the response body is always read regardless of the status returned by the server.

Reading the JSON response

HttpEntity entity = response.getEntity();

**Release the Connection**

This is a crucial step to keep things flowing. We must tell *HttpClient* that we are done with the connection and that it can now be reused. Without doing this *HttpClient* will wait indefinitely for a connection to free up so that it can be reused.

method.releaseConnection();

**Deal with the Repsonse**

We've now completed our interaction with *HttpClient* and can just concentrate on doing what we need to do with the data.

It's worth noting that if you were retrieving the response as a stream and processing it as it is read, this step would actually be combined with reading the connection, and when you'd finished processing all the data, you'd then close the input stream and release the connection.

entity.getContent();

**Validate the Objects inside the Object.**

This process involves

1. Converting the JSON response to JSON Objects

The response got after the execution is converted into JSON Objects so that the objects and their values can be validated for every request. The JSON string response is de serialized to JSON objects.

1. Using the JSONAsserter class methods the JSON objects can be validated for its values.

A Sample JSON response would look like

{  
   "success":true,  
   "userPermissions":{  
      "username":"admin",  
      "userId":1,  
      "permissions":{  
         "Processing\_Tab":1,  
         "Search\_Review\_Export\_Tab":2,  
         "Reports\_Tab":4,  
         "Admin\_Tab":8  
      },  
      "tabPermission":15  
   },  
   "restartInProgress":null,  
   "nextAction":{  
      "tab":"dashboard"  
   }  
}

Here the top level objects are success, userPermissions, restartInProgress, nextAction and the sublevel objects are username, userId, permissions, tabPermission, etc.

Examples of validations are

1. Success is true.
2. userPermission/username is admin.
3. nextAction is dashboard.

These can be validated through the method

assertJSONObjects(String jsoncontent, String path, String value)

jsoncontent 🡪 the JSON response.

Path 🡪 The object path that needs to be followed to get the value.

Eg: userPermission/username

Value 🡪 The value against which the assertion is done.

**Test Flow**

1. All http Tests needs to extend the httpjsonAppTest.
2. The httpjsonAppTest would
   1. Initialize cookies.
   2. Store cookies for further use.
3. In the test method
   1. Call the getJSONFromServer method to perform the http execute with appropriate parameters.
   2. Pass the JSON response that is returned to assertJSONObjects method to validate the JSON objects and its values.
4. Stop the test
   1. Call the StopHttpTest method that clears the cookiestore.

**Sample Test Case**

**public** **class** HttpTest **extends** HttpJsonAppTest {

@BeforeMethod(alwaysRun= **true**)

**public** **void** startTest() **throws** Exception {

**super**.inithttpTest();

}

@Test(enabled = **true**)

**public** **void** testsignin() **throws** ClientProtocolException, URISyntaxException, IOException, SecurityException, IllegalArgumentException, NoSuchFieldException, IllegalAccessException{

String jsonContent = HttpClientDriver.*getJSONFromServer*("POST", 8080, "/auth/signIn", "username=admin&password=hello");

JSONAsserter.*assertJSONObjects*(jsonContent, "userPermissions/username", "admin");

jsonContent = HttpClientDriver.*getJSONFromServer*("POST", 8080, "/userGroup/listForUserView", "id=");

JSONAsserter.*assertJSONSuccessObjectTrue*(jsonContent);

}

@AfterMethod(alwaysRun= **true**)

**public** **void** stopTest() **throws** Exception {

**super**.stophttpTest();

}

}