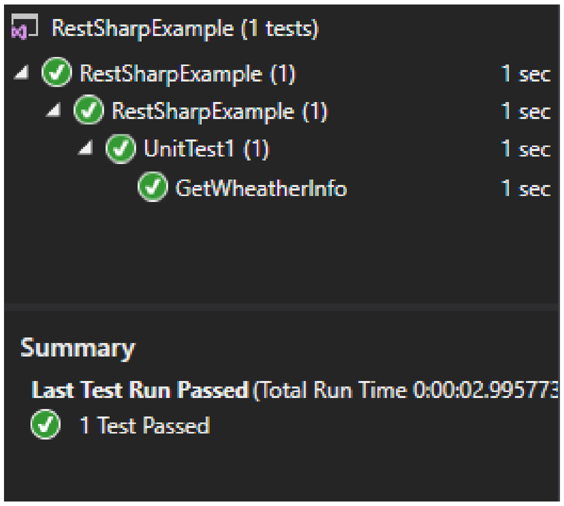
How to Write REST API Test using RestSharp?

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using RestSharp;

namespace RestSharpExample
{
    [TestClass]
    public class Weather
    {
        [TestMethod]
        public void GetWeatherInfo()
        {
      //Creating Client connection
            RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");

      //Creating request to get data from server
            RestRequest restRequest = new RestRequest("Guntur",Method.GET);

      // Executing request to server and checking server response to the it
            IRestResponse restResponse = restClient.Execute(restRequest);

            // Extracting output data from received response
      string response = restResponse.Content;

      // Verifiying reponse
      if(!response.contains("Guntur")
               Assert.Fail("Whether information is not displayed");
        }
    }
}
```

The above code will produce the same response which was received when opened the same URL on browser.



```
RestSharpExample (1 tests)
◢ RestSharpExample (1)            1 sec
   ◢ RestSharpExample (1)         1 sec
      ◢ UnitTest1 (1)             1 sec
            GetWheatherInfo       1 sec

Summary
 Last Test Run Passed (Total Run Time 0:00:02.995773
  1 Test Passed
```

Understanding the Code

CODE SEGMENT 1: Creating Client Connection

```
RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");
(Or)
const string baseURL = "http://restapi.demoqa.com/utilities/weather/city/";
```

```
RestClient restClient = new RestClient();
restClient.BaseUrl =new Uri(baseURL);
```

To perform any *API* call we require to open a browser and provide *URL* in address bar. Creation of *RestClient* in *RestSharp* is similar to that of opening a browser and setting it ready to call any server using *URL*. *RestClient* initializes a client with the specified *Base URI* to send requests. In our case the *Base URI* is *http://restapi.demoqa.com/utilities/weather/city*. This is called the *Base URI* because it is root address of the resource. Adding ***/Guntur*** at the end appends the exact resource name in the *URI* that we are trying to access.

CODE SEGMENT 2: Creating Request from Client to Server

```
RestRequest restRequest = new RestRequest("Guntur",Method.GET);
```

As client is created, it is ready to send any request to server. So, request has to be create as specified. ***RestRequest*** class creates *HTTP Requests* against specified *URL* with type of ***Protocol***. The above request specifies that it needs to get the information from the URL(*Base URL+ "Guntur"*)
RestRequest supports creating Request of different HTTP method types (GET, POST, PUT, PATH, DELETE, UPDATE, HEAD and OPTIONS)

CODE SEGMENT 3: Execute Request on Server

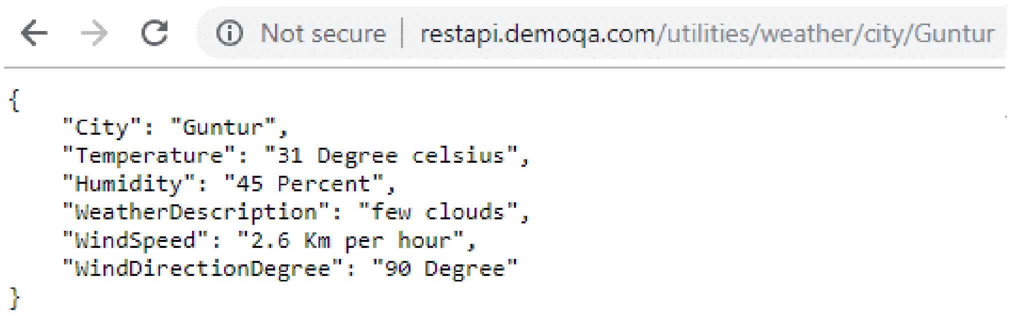
```
IRestResponse restResponse = restClient.Execute(restRequest);
string response = restClient.Execute(restRequest).Content;
```

Now that *RestRequest* object is there, we execute our request to get the resource response from the server. ***Execute*** method of *RestRequest* object actually sends the request to the remote server and gets a response back. This is why the return type of the *request.Execute()* is specified as *IRestResponse*.
IResponse interface represents a *Response* returned from a server. This *Response* object will contain all the data sent by the server. Different method can be called on the *Response* object to get different parts. For e.g. call to get ***Response Headers, Response Status and the Response Body.***

CODE SEGMENT 4: Verifying Response

```
if(!response.contains("Guntur"){
    Assert.Fail("Whether information is not displayed");
}
```

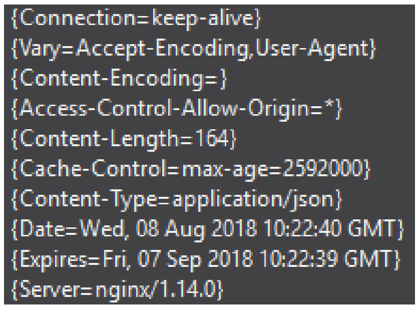
Assert class is from *Unit Framework* is used to verify various assertions. In our example we are checking that weather information is provided as response from the server for given city. So if response contains city name as *Guntur* which confirms proper response is received from server



```
{
    "City": "Guntur",
    "Temperature": "31 Degree celsius",
    "Humidity": "45 Percent",
    "WeatherDescription": "few clouds",
    "WindSpeed": "2.6 Km per hour",
    "WindDirectionDegree": "90 Degree"
}
```

How to read different Header Types from HTTP Response?

The Response interface provides direct parameter property collection "Headers" to access individual header or all the Headers. Below image shows the available list parameters.



```
{Connection=keep-alive}
{Vary=Accept-Encoding,User-Agent}
{Content-Encoding=}
{Access-Control-Allow-Origin=*}
{Content-Length=164}
{Cache-Control=max-age=2592000}
{Content-Type=application/json}
{Date=Wed, 08 Aug 2018 10:22:40 GMT}
{Expires=Fri, 07 Sep 2018 10:22:39 GMT}
{Server=nginx/1.14.0}
```

```
Dictionary<string, string> HeadersList = new Dictionary<string, string>();
 /*Key*//*Value*/
```

As we get headers list parameters which is a collection. We iterate through each list item, which contains header name and value, so we split the parameters and add each into the dictionary created.

```
[TestMethod]
public void GetContentTypeHeader()
{
 RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");
 RestRequest restRequest = new RestRequest("Guntur", Method.GET);
 IRestResponse restResponse = restClient.Execute(restRequest);
 // Creating a dictionary and adding all headers and their values
 Dictionary<string, string> HeadersList = new Dictionary<string, string>();
          foreach (var item in restResponse.Headers)
          {
            string[] KeyPairs=item.ToString().Split('=');
              HeadersList.Add(KeyPairs[0],KeyPairs[1]);
 }
          //Asserting content type from header
          Assert.AreEqual("application/json", HeadersList["Content-Type"],"Content-type not matching");


          //Asserting Server name from header
          Assert.AreEqual("nginx/1.14.0", HeadersList["Server"]," Server not matching");
     }
```

Validate Response Status using RestSharp

This tutorial will cover the following verification for ***Response Status*** :

- ***Validating Response Status Code***
- ***Validating Error Status Code***
- ***Validating Response Status Line***

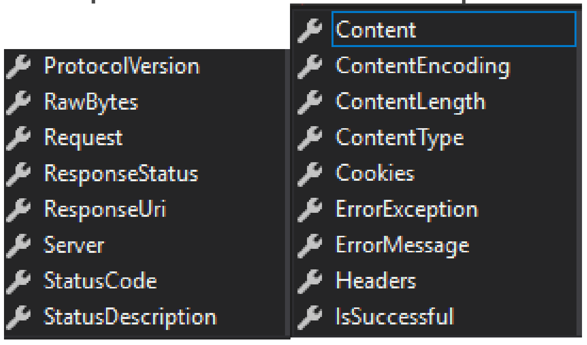
```
[TestMethod]
public void GetWheatherInfo()
{
RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");
RestRequest restRequest = new RestRequest("Guntur",Method.GET);
IRestResponse restResponse = restClient.Execute(restRequest);
string response = restResponse.Content;
if(!response.contains("Guntur") Assert.Fail("Whether information is not displayed");
}
```

IResponse object which represents the HTTP Response packet received from the Web service Server. ***HTTP Response*** *contains* ***Status****, collection of* ***Headers*** *and a* ***Body***. Hence, it becomes obvious that the Response object should provide mechanisms to read Status, Headers and Body.

How to Validate Response Status Code?

IResponse is an interface which has lots of properties, mostly which can help to get parts of the received response. Look at some of the important properties. A simple response followed by a dot (***restResponse***) would show the available properties on the response object. As shown in the image below.

IResponse Interface Properties



As noticed above, ***StatusCode*** can be used to get the status code of the *REST Response*. This property returns an ***HttpStatusCode*** which is an ***Enum.***

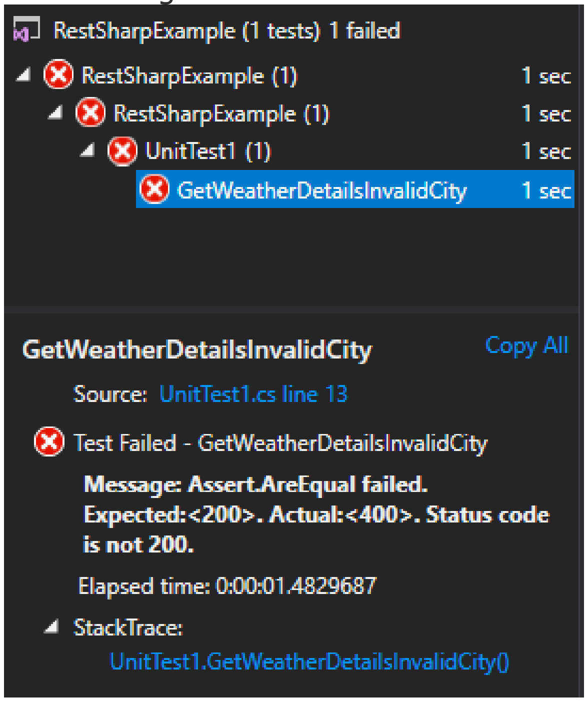
```
[TestMethod]

public void GetWheatherInfo()

{

RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");

RestRequest restRequest = new RestRequest("Guntur", Method.GET);

IRestResponse restResponse = restClient.Execute(restRequest);

//Extract status code from received response and store in an Interger

int StatusCode = (int)restResponse.StatusCode;

//Assert that correct Status is returned

Assert.AreEqual (200, StatusCode, "Status code is not 200");

Or Assert.AreEqual (HttpStatusCode.ok, StatusCode, "Status code is not 200");

}
```

How to Validate an Error Status Code?

Status Codes returned by the Server depends on whether the ***HTTP Request*** was successful or not. If the *Request* is successful, *Status Code 200* is returned otherwise *Status Code* other than 200 will be returned.

```
[TestMethod]

public void GetWeatherDetailsInvalidCity()

{

RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");
/*Invalid city*/

RestRequest restRequest = new RestRequest("abcdef", Method.GET);

IRestResponse restResponse = restClient.Execute(restRequest);


int StatusCode = (int)restResponse.StatusCode;

Assert.AreEqual (200,      StatusCode , "Status code is not 200");

}
```

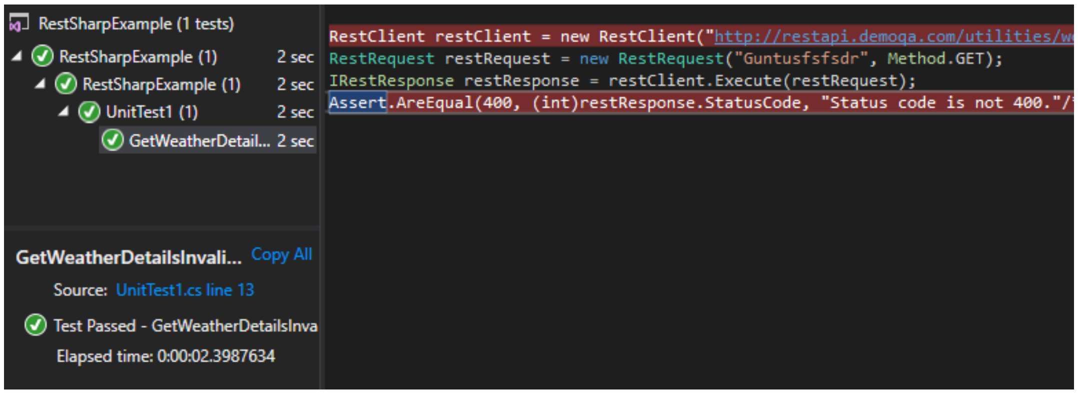
Now run your test class and check the result. Our negative test *GetWeatherDetailsInvalidCity* will fail. The reason is that the web service returns an error code of 400 when invalid city name is sent to it. However, the test is validating for expected value of 200 as shown in the image below.



```
RestSharpExample (1 tests) 1 failed
  RestSharpExample (1)                     1 sec
    RestSharpExample (1)                   1 sec
      UnitTest1 (1)                        1 sec
        GetWeatherDetailsInvalidCity       1 sec

GetWeatherDetailsInvalidCity                        Copy All
    Source: UnitTest1.cs line 13
  Test Failed - GetWeatherDetailsInvalidCity
     Message: Assert.AreEqual failed.
     Expected:<200>. Actual:<400>. Status code
     is not 200.
     Elapsed time: 0:00:01.4829687
  StackTrace:
       UnitTest1.GetWeatherDetailsInvalidCity()
```

A quick change in the expected result in the code will make sure that this test passes. Update respective line as in the code below:

Assert.AreEqual (400, StatusCode, "Status code is not 200");
Assert.AreEqual (HttpStatusCode.BadRequest, StatusCode, "Status code is not 200");



RestSharpExample (1 tests)
RestSharpExample (1) 2 sec
RestSharpExample (1) 2 sec
UnitTest1 (1) 2 sec
GetWeatherDetail... 2 sec

GetWeatherDetailsInvali... Copy All
Source: UnitTest1.cs line 13
Test Passed - GetWeatherDetailsInva
Elapsed time: 0:00:02.3987634

```
RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/w
RestRequest restRequest = new RestRequest("Guntusfsfsdr", Method.GET);
IRestResponse restResponse = restClient.Execute(restRequest);
Assert.AreEqual(400, (int)restResponse.StatusCode, "Status code is not 400."/
```

How to Validate Response Status Description, Protocol version?

Just as we get the *Status Code* as mentioned above, we can get the ***Status Description & Protocol Version*** as well. To get these properties we will simply call the *Response.PropertyName*.

```
string statusDescription = restResponse.StatusDescription;

string protocolVersion = restResponse.ProtocolVersion.ToString();
```

Check if a String is contained in the Response Body

ResponseBody can return the response body in a String format. We can use simple String methods to verify certain basic level of values in the Response. For e.g. we can use the String.contains() method to see if the Response contains a "Guntur" in it. The below code shows how to check for sub string presence.

```
using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using RestSharp;


namespace RestSharpExample

{

    [TestClass]

    public class Whether

    {

        [TestMethod]

        public void GetWheatherInfo()

        {
```

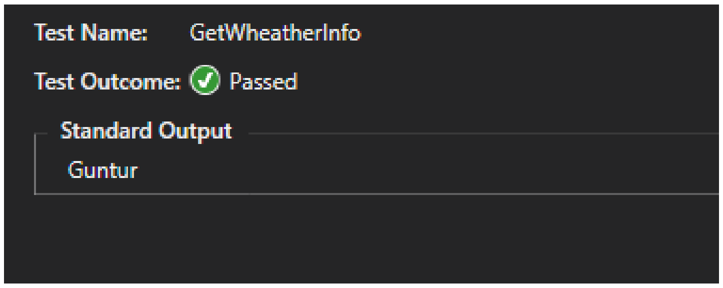
```
    //Creating Client connection

         RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");


    //Creating request to get data from server

        RestRequest restRequest = new RestRequest("Guntur",Method.GET);


    // Executing request to server and checking server response to the it

         IRestResponse restResponse = restClient.Execute(restRequest);

    // Extracting output data from received response

       string response = restResponse.Content;

    // Verifiying reponse

   JObject Jobj = JObject.Parse(response);

    Assert.AreEqual ("Guntur", Jobj["city"], "City is not Guntur");

       }

   }

}
```

Check String presence by ignoring alphabet casing

Response interface gives you a mechanism to extract nodes based on a given Json using Jobject.Parse. There is a method calledJObject.Parse(restResponse.Content), which returns an element-node Newtonsoft.Json.Linq.JObject. This object can be used to further query specific parts of the Response Json.

```
      [TestMethod]

      public void GetWheatherInfo()

      {

    //Creating Client connection

          RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");

    //Creating request to get data from server

          RestRequest restRequest = new RestRequest("Guntur",Method.GET);

    // Executing request to server and checking server response to the it

         IRestResponse restResponse = restClient.Execute(restRequest);

     // Extracting output data from received response

    string response = restResponse.Content;
    // Parsing JSON content into element-node JObject
          var jObject = JObject.Parse(restResponse.Content);
      //Extracting Node element using Getvalue method
           string city=jObject.GetValue("City").ToString();
    // Let us print the city variable to see what we got
    Console.WriteLine("City received from Response " + city);
   // Validate the response
   Assert.AreEqual(city, "Guntur", "Correct city name received in the Response");
}
```

The output of the code passes the assertion and it also prints the City name retrieved from the Response. As shown in the image below



```
Test Name:      GetWheatherInfo
Test Outcome:   Passed
Standard Output
  Guntur
```

CityExtracted

On the similar lines you can extract any part of the Json response using the JObject implementation of RestSharp. This is very convenient, compact and easy way to write tests.

Sample Code to read all the nodes from Weather API Response

Now that we know how to read a node using JObject, here is a small piece of code that reads all the nodes and prints them to the Console.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json.Linq;
using RestSharp;

namespace RestSharpExample
{
   [TestClass]
   public class UnitTest1
```

```
    {
        [TestMethod]
        public void DisplayAllNodesInWeatherAPI ()
        {
                    RestClient restClient = new RestClient("http://restapi.demoqa.com/utilities/weather/city/");
                     RestRequest restRequest = new RestRequest("Guntur", Method.GET);
                     IRestResponse restResponse = restClient.Execute(restRequest);

 //Parse the JSON response

                         var jObject = JObject.Parse(restResponse.Content);

     //Print City
               Console.WriteLine(jObject.GetValue("City");

                    //Print Temperature
               Console.WriteLine(jObject.GetValue("Temperature");

                  //Print Humidity
                  Console.WriteLine(jObject.GetValue("Humidity");

                  //Print Weather
               Console.WriteLine(jObject.GetValue("Weather");

                  //Print WindSpeed
               Console.WriteLine(jObject.GetValue("WindSpeed");

                  //Print WindDirectionDegree
               Console.WriteLine(jObject.GetValue("WindDirectionDegree");

        }
    }
}
```

How to make a POST Request using Rest Sharp?

Let us now try to test the above web service. During this tutorial we will learn about

- *POST request using Rest Sharp*
- *Creating Json data using Simple Json library*
- *Sending JSON content in the body of Request*
- *Validating the Response.*

In order to create ***JSON objects*** in the code we will add Newtonsoft.Json.Linq library in the class path. You can download Newtonsoft.Json.Linq from NuGet using
Let us begin step by step with the code

Step 1: Create a Request pointing to the Service Endpoint

```
RestClient restClient = new RestClient("http://restapi.demoqa.com/customer");
```

This code is self explanatory. If you are not able to understand it you have missed the earlier tutorials in this series, please go through them first.

Step 2: Create a JSON request which contains all the fields

```
JObject jObjectbody = new JObject();
            jObjectbody.Add("FirstName", "Narayn");
            jObjectbody.Add("LastName", "Kaluri");
            jObjectbody.Add("UserName", "NaraynKaluri");
            jObjectbody.Add("Password", "Password@123");
            jObjectbody.Add("Email", "abc@hotmail.com");
```

JObject is a class that is present in Newtonsoft.Json.Linq. namespace. This class is a programmatic representation of a ***JSON*** string. Take a look at the **Request JSON** above for our test web service, you will notice that there are multiple nodes in the **Json**. Each node can be added using the ***JObject.Add(String, String)*** method. First parameter is json propertyName and second is propertyValue.

Step 3: Add JSON body in the request and send the Request

```
RestRequest restRequest = new RestRequest("Guntur", Method.POST);

restRequest.AddParameter("application/json",jObjectbody,ParameterType.RequestBody);
```

This web service accepts a **JSON** body. By this step we have created our **JSON**body that needs to be sent. In this step we will simply add the **JSON** object to the body of the **HTTP Request** and make sure that the ***Content-Type*** header field has a value of ***application/json***. You can put the **JSON** object in the body using the method called ***restRequest.AddParameter(content type, object, parameter type).*** This method lets you updated the content of ***HTTP Request*** Body. However, if you call this method multiple times the body will be updated to the latest **Json** object.

```
//Specifies request content type as Json
restRequest.RequestFormat =DataFormat.Json;

//Create a body with specifies parameters as json
```

```
restRequest.AddBody(new { FirstName = "Narayn", LastName= "Kaluri", UserName= "NarsaynKaluri" ,
Password= "Passwvvord@123" , Email= "avbc@hotmail.com" });
```

Step 4: Validate the Response

Once we get the response back, all we have to do is validate parts of the response. Let us validate Success Code text in the Response, we will be using ***JObject***.

Now that we have sent the Request and received a Response, let us validate ***Status Code*** and ***Response Body*** content. This is similar to validation techniques that we discussed in previous tutorials. I would strongly suggest that you read those tutorials. The above code does the validation, it is self-explanatory.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json.Linq;
using RestSharp

namespace RestSharpExample
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void PostMethod()
        {
         RestClient restClient = new RestClient("http://restapi.demoqa.com/customer");

            //Creating Json object
            JObject jObjectbody = new JObject();
            jObjectbody.Add("FirstName", "Narayn");
            jObjectbody.Add("LastName", "Kaluri");
            jObjectbody.Add("UserName", "NaraynKasdfsdfluri");
            jObjectbody.Add("Password", "Passwovasdfsdaf23");
            jObjectbody.Add("Email", "abcdafsad@hotmail.com");

         RestRequest restRequest = new RestRequest("/register" ,Method.POST);

         //Adding Json body as parameter to the post request
         restRequest.AddParameter("application/json",jObjectbody,ParameterType.RequestBody);

         IRestResponse restResponse = restClient.Execute(restRequest);

         Assert.Contains("OPERATION_SUCCESS ", restRequest.Content," Post failed ");
        }
    }
}
```

Changing the HTTP Verb on a POST request

One of the key aspect of Web service testing is to verify negative scenarios on the Endpoint. There could be many negative scenarios, some of them are

- *Sending incomplete POST Data*
- *Sending Json data with incorrect syntax*
- *Sending incorrect Verb in the Request.*

Let us see what the impact will be if we send a GET request to an Endpoint that expects POST. Below code tries to do that, we will just print out the Response status code and the Response body to see if we get any error.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json.Linq;
using RestSharp

namespace RestSharpExample
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void PostMethod()
        {
         RestClient restClient = new RestClient("http://restapi.demoqa.com/customer");

           //Creating Json object
            JObject jObjectbody = new JObject();
            jObjectbody.Add("FirstName", "Narayn");
            jObjectbody.Add("LastName", "Kaluri");
            jObjectbody.Add("UserName", "NaraynKasdfsdfluri");
            jObjectbody.Add("Password", "Passwovasdfsdaf23");
            jObjectbody.Add("Email", "abcdafsad@hotmail.com");

         RestRequest restRequest = new RestRequest("/register" ,Method.GET);

         //Adding Json body as parameter to the post request
         restRequest.AddParameter("application/json",jObjectbody,ParameterType.RequestBody);

         IRestResponse restResponse = restClient.Execute(restRequest);

         Assert.Contains("OPERATION_SUCCESS ", restRequest.Content," Post failed ");
        }
    }
}
```

Basic Authentication Flow

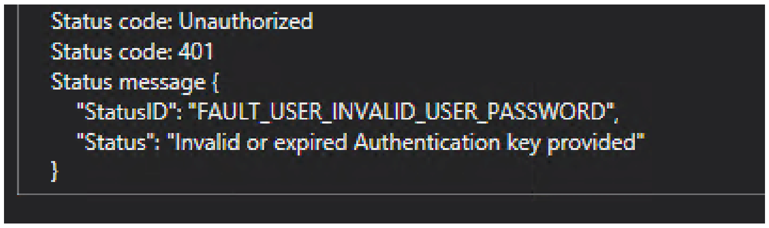
A REST request can have a special header called ***Authorization Header,*** this header can contain the credentials (username and password) in some form. Once a request with Authorization Header is received, server can validate the credentials and can let you access the private resources.

Endpoint: http://restapi.demoqa.com/authentication/CheckForAuthentication

```
[TestMethod]
public void AuthenticationBasics()
{
RestClient restClient = new RestClient();
restClient.BaseUrl = new Uri("http://restapi.demoqa.com/authentication/CheckForAuthentication");
RestRequest restRequest = new RestRequest(Method.GET);
IRestResponse restResponse = restClient.Execute(restRequest);
 Console.WriteLine("Status code: " + restResponse.StatusCode);
 Console.WriteLine ("Status message " + restResponse.Content);
}
```

In the code above we are simply making a ***HTTP GET*** request to the endpoint. In this code we have not added any **Authorization** header. So the expected behaviour is that we will get Authorization error. If you run this test, you will get following output.

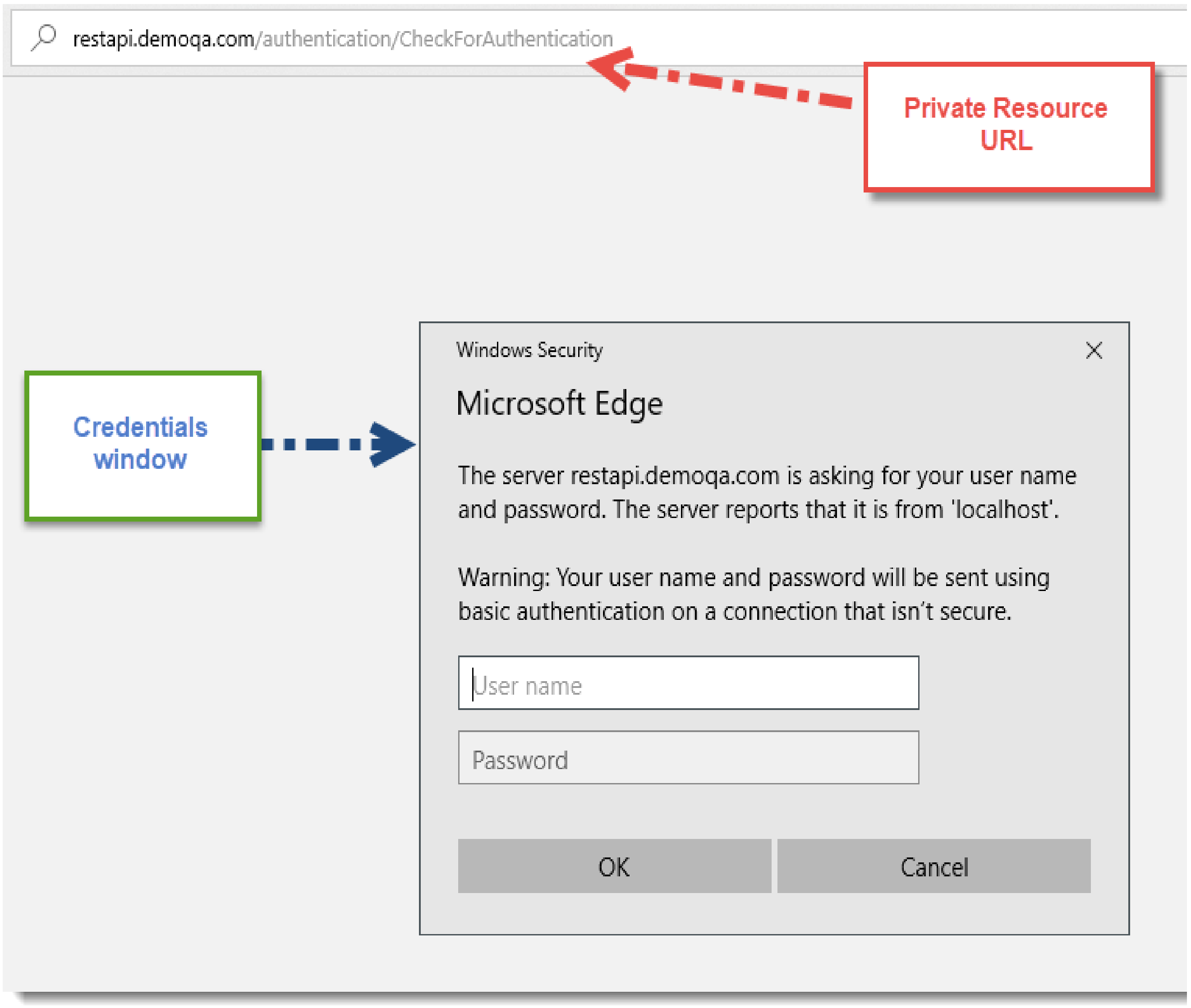
```
Status code: 401
Status message:

{

“StatusID”: “FAULT_USER_INVALID_USER_PASSWORD”,

“Status”: “Invalid or expired Authentication key provided”

}
```



```
Status code: Unauthorized
Status code: 401
Status message {
    "StatusID": "FAULT_USER_INVALID_USER_PASSWORD",
    "Status": "Invalid or expired Authentication key provided"
}
```

The output clearly says that we have ***“Invalid or expired Authentication key provided”*** error. Which means that either there was no Authentication information or the information supplied was invalid. Eventually, server denies our request and returns an Error response.
Note: *Pay special attention to the* ***Status code*** *returned. In case of* ***Authentication*** *failures Server should respond with a status code of* ***401 Unauthorized.***

Try to hit that URL using a browser. You should get a Username and Password prompt. Below image shows what you should be getting when you hit this URL from browser.



The server restapi.demoqa.com is asking for your user name and password. The server reports that it is from 'localhost'.

Warning: Your user name and password will be sent using basic authentication on a connection that isn't secure.

What is Authorization and How does Authorization works in REST WebServices

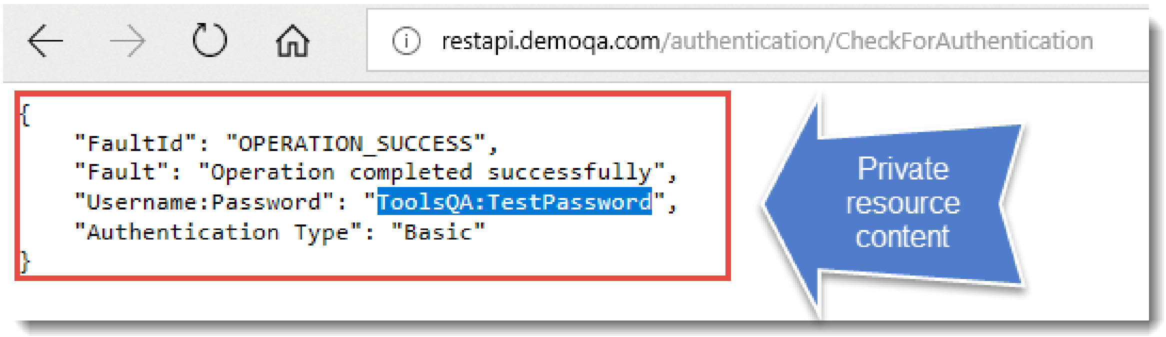
Authorization

Authorization is the process of giving access to someone. If you are Authorized then you have access to that resource. Now to Authorize you need to present credentials and as we discussed earlier that process is called Authentication. Hence Authorization and Authentication are closely related terms and often used interchangeably.

Before ending the tutorial let us see the contents of the private resource in the URL mentioned above. To do that enter the following credentials

Username: ToolsQA
Password: TestPassword

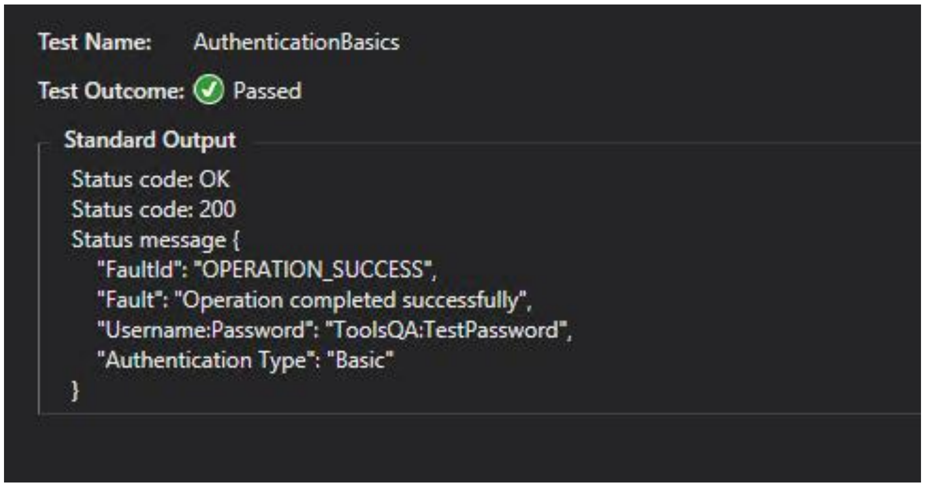
Server will be able to **Authenticate** and then **Authorize** you to access the private resource content. Below image shows the content after successful Authentication.



```
{
    "FaultId": "OPERATION_SUCCESS",
    "Fault": "Operation completed successfully",
    "Username:Password": "ToolsQA:TestPassword",
    "Authentication Type": "Basic"
}
```

With this basic understanding of Authentication and Authorization, read the coming tutorials where we will discuss the specific types of **Authentication** models in **REST API**.

```
    [TestMethod]
    public void AuthenticationBasics()
    {
        RestClient restClient = new RestClient();

        restClient.BaseUrl = new Uri("http://restapi.demoqa.com/authentication/CheckForAuthentication");

        RestRequest restRequest = new RestRequest(Method.GET);

        restClient.Authenticator = new HttpBasicAuthenticator("ToolsQA", "TestPassword");
request.AddHeader("Content-Type", "application/x-www-form-urlencoded");

        IRestResponse restResponse = restClient.Execute(restRequest);
        Console.WriteLine("Status code: " + (int)restResponse.StatusCode);
        Console.WriteLine("Status message " + restResponse.Content);
    }
```



```
Test Name:     AuthenticationBasics
Test Outcome:  Passed
  Standard Output
   Status code: OK
   Status code: 200
   Status message {
     "FaultId": "OPERATION_SUCCESS",
     "Fault": "Operation completed successfully",
     "Username:Password": "ToolsQA:TestPassword",
     "Authentication Type": "Basic"
   }
```

```
     [TestMethod]
    public void AuthenticationBasics()
```

```
{
   RestClient restClient = new RestClient();

   const string CertificatePath = "C:\\Certificate.cert";

   X509Certificate x509Certificate = new X509Certificate(CertificatePath,"CertPassword");

   restClient.ClientCertificates.Add(x509Certificate);

   restClient.BaseUrl = new Uri("http://Sample.com");

   RestRequest restRequest = new RestRequest(Method.GET);

   restClient.Authenticator = new HttpBasicAuthenticator("User", "TestPassword");

   IRestResponse restResponse = restClient.Execute(restRequest);
   Console.WriteLine("Status code: " + (int)restResponse.StatusCode);
   Console.WriteLine("Status message " + restResponse.Content);
}
```

Measure HTTP Response Time from a REST Client

```
DateTime T = System.DateTime.UtcNow;//--> Note the initial Time

            HttpWebResponse response = (HttpWebResponse) req.GetResponse();
            // Get the stream containing content returned by the server.
            dataStream = response.GetResponseStream();
            // Open the stream using a StreamReader for easy access.
        StreamReader reader = new StreamReader(dataStream);
    // Read the content.
    string responseFromServer = reader.ReadToEnd();
TimeSpan TT = System.DateTime.UtcNow – T;
resposnseTime=TT.TotalMilliseconds ;//--> Note the Time Difference
```

How to use OAuth2 in RestSharp

```
var client = new RestClient("http://example.com/myapi/oauth/token");
RestRequest request = new RestRequest() { Method = Method.POST };

request.AddHeader("Content-Type", "application/json");
request.AddParameter("grant_type", "client_credentials");
request.AddParameter("client_id", "client-app");
request.AddParameter("client_secret", "secret");

var response = client.Execute(request);
```

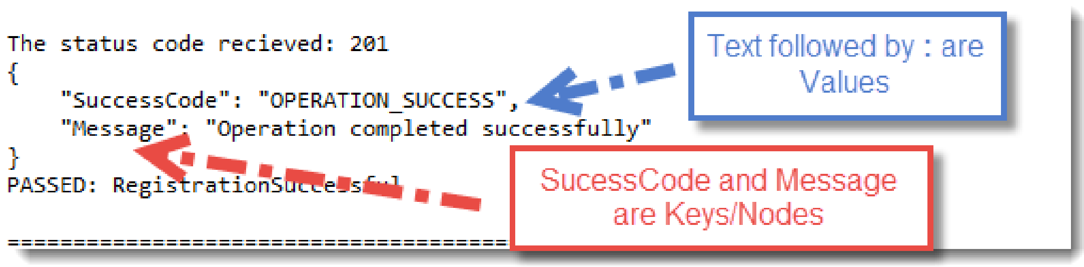
How to Deserialize JSON Response to Class with Rest Sharp?

Let us continue with our previous chapter ***Making a POST request using Rest-Sharp.*** In the previous chapter we made a successful post request and the following **Response body** was received.
We used ***JObject.Parse*** to validate parts of the Response body. Know we will convert this Response body into a Class. Representing a JSON, or any structured data including XML, into a programming class is

called ***Deserialization*** of JSON. The term ***Deserialization*** here means the conversion from String form of JSON to a Class form. This is also called ***Object Representation*** of structured data. Here structured data is the JSON.
In order to convert the *JSON* into a Class representation, we will first create a Class that has all the nodes present in the *JSON*. The Success response above has two nodes

- ***SuccessCode***
- ***Message***

The value of these two nodes is ***String***. Below image shows the parts of the response.



```
The status code recieved: 201
{
    "SuccessCode": "OPERATION_SUCCESS",
    "Message": "Operation completed successfully"
}
PASSED: RegistrationSuccessful

======================================
```

Text followed by : are Values
SucessCode and Message are Keys/Nodes

```
public class ResponseData

   {

       // Succes code of response received

       public string SuccessCode;

       // Message of response received

       public string Message;

   }
```

Now we have a ***ResponseData*** class that can store all the nodes present in Success Response. Let us understand how we can use **RestSharp** to automatically convert Response Body Json to the instance of the ***ResponseData*** class.
By using deserialize object method of simplejson class, we can get all element nodes from string json to required format which in our case is a class(ResponseData).

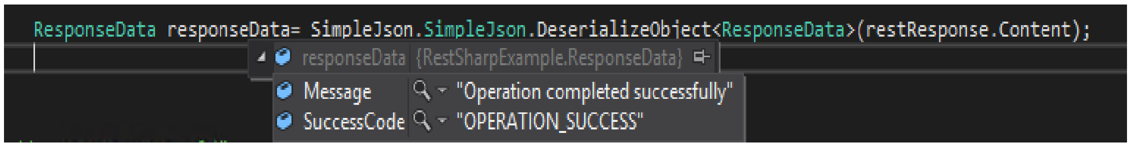
SimpleJson.DeserializeObject<Class>(JSon string);

```
ResponseData responseData=

SimpleJson.DeserializeObject<ResponseData>(restResponse.Content);
```

This method call will internally do two things

- *Create an instance of **ResponseData***
- *Then copy the Node value of Json to the respective variable in the Class. For e.g. **Value of SuccessCode**node to the **Variable SucessCode** in the class.*

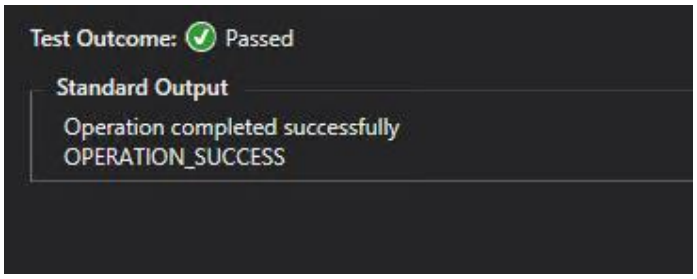
To visualize it, let us look at the image below.



```
ResponseData responseData= SimpleJson.SimpleJson.DeserializeObject<ResponseData>(restResponse.Content);
responseData {RestSharpExample.ResponseData}
Message "Operation completed successfully"
SuccessCode "OPERATION_SUCCESS"
```

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Newtonsoft.Json.Linq;
using RestSharp

namespace RestSharpExample
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void PostMethod()
        {
         RestClient restClient = new RestClient("http://restapi.demoqa.com/customer");

            //Creating Json object
            JObject jObjectbody = new JObject();
            jObjectbody.Add("FirstName", "Narayn");
            jObjectbody.Add("LastName", "Kaluri");
            jObjectbody.Add("UserName", "NaraynKasdfsdfluri");
            jObjectbody.Add("Password", "Passwovasdfsdaf23");
            jObjectbody.Add("Email", "abcdafsad@hotmail.com");

         RestRequest restRequest = new RestRequest("/register" ,Method.POST);

          //Adding Json body as parameter to the post request
          restRequest.AddParameter("application/json",jObjectbody,ParameterType.RequestBody);

          IRestResponse restResponse = restClient.Execute(restRequest);
         ResponseData responseData=SimpleJson.DeserializeObject<ResponseData>(restResponse.Content);


          Console.WriteLine(responseData.Message);
          Console.WriteLine(responseData.SuccessCode);
        }
    }
    public class ResponseData
    {
        // Succes code of response received
        public string SuccessCode;
        // Message of response received
        public string Message;
    }
```

Output for above sample code is as shown below.



```
Test Outcome: Passed
Standard Output
Operation completed successfully
OPERATION_SUCCESS
```

Now you can use the responseBody variable to apply any assertions that you want or may be pass it as an input to other tests.

```
              Assert.AreEqual("OPERATION_SUCCESS",
responseData.SuccessCode);
              Assert. AreEqual ("Operation completed successfully",
responseData.Message);
```

Deserializing Response body based on Response Status code

In real life it is possible for a REST Api endpoint to return multiple Json formats. For e.g. the API in this tutorial can return a completely different response in case of Failures. Below is the Json that is returned in failure cases.

```
{
```

```
“FaultId”: “User already exists”,

“fault”: “FAULT_USER_ALREADY_EXISTS”

}
```

If we try to ***Deserialize*** this response into ***ResponseData,*** it will not work. The reason is that while **Deserializing *RestSharp*** will not be able to find **SuceessCode** and **Message** fields in the ***Response*** body. As a result the two variables will be set to null.
So how do we handle these cases? This particular case of Success and Failure can be easily catered using the returned ***Status code***. API in this tutorial returns **201** status code in case of Success and **200** in case of Error. We will also need another class that represents the failure Json Response.
We can see that there are two nodes that are present in **Failure** cases.

- ***FaultId***
- ***Fault***

```
public class RegistrationFailureResponse

{

 public string FaultId;

 public string fault;

}
```

To determine which scenario to execute we can simply put an if check in the Test code. Below is the updated Test code.

```
using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Newtonsoft.Json.Linq;

using RestSharp


namespace RestSharpExample

{

    [TestClass]

    public class UnitTest1

    {

        [TestMethod]

        public void PostMethod()

        {
```

```
        RestClient restClient = new RestClient("http://restapi.demoqa.com/customer");

            //Creating Json object
          JObject jObjectbody = new JObject();
          jObjectbody.Add("FirstName", "Narayn");
          jObjectbody.Add("LastName", "Kaluri");
          jObjectbody.Add("UserName", "NaraynKasdfsdfluri");
          jObjectbody.Add("Password", "Passwovasdfsdaf23");
          jObjectbody.Add("Email", "abcdafsad@hotmail.com");

          RestRequest restRequest = new RestRequest("/register" ,Method.POST);

          //Adding Json body as parameter to the post request
          restRequest.AddParameter("application/json",jObjectbody,ParameterType.RequestBody);

          IRestResponse restResponse = restClient.Execute(restRequest);

          ResponseData responseData=SimpleJson.DeserializeObject<ResponseData>(restResponse.Content);

          if((int)restResponse.StatusCode==200)
   {

           // Deserialize the Response body into RegistrationFailureResponse
           FailureResponse responseBody =SimpleJson.DeserializeObject<FailureResponse>(restResponse.Content);

           // Use the FailureResponse class instance to Assert the values of Response.

Assert.AreEquals("User already exists", responseBody.FaultId);
Assert. AreEquals ("FAULT_USER_ALREADY_EXISTS", responseBody.fault);
}

else if ((int)restResponse.StatusCode==201)
{
       // Deserialize the Response body into SuccessResponse
       SuccessResponse responseBody =SimpleJson.DeserializeObject< SuccessResponse >(restResponse.Content);
```

```
        // Use the SuccessResponse class instance to Assert the values of Response.

        Assert.assertEquals("OPERATION_SUCCESS", responseBody.SuccessCode);

        Assert.assertEquals("Operation completed successfully", responseBody.Message);

 }

}

   public class SuccessResponse

   {

       // Succes code of response received

       public string SuccessCode;

       // Message of response received

       public string Message;

   }


   public class FailureResponse

   {

       // FaultId of response received

       public string FaultId;

       // Fault of response received

       public string Fault;

}
```

```
// execute the request
IRestResponse response = client.Execute(request);
var content = response.Content; // raw content as string

// or automatically deserialize result
// return content type is sniffed but can be explicitly set via RestClient.AddHandler();
IRestResponse<Person> response2 = client.Execute<Person>(request);
var name = response2.Data.Name;

// or download and save file to disk
client.DownloadData(request).SaveAs(path);

// easy async support
client.ExecuteAsync(request, response => {
    Console.WriteLine(response.Content);
});

// async with deserialization
var asyncHandle = client.ExecuteAsync<Person>(request, response => {
    Console.WriteLine(response.Data.Name);
});

// abort the request on demand
asyncHandle.Abort();
```

Basic usage of RestSharp

```
var client = new RestClient ("URLName");

var request = new RestRequest("Resource/{id}", Method.POST);
request.AddParameter("name", "Yourvalue");
request.AddUrlSegment("id", 123);
request.AddHeader("header", "value");
RestResponse response = client.Execute(request);
```

New RestRequest will create RestRequest to our specified URL.

request.AddParameter will add parameter to your request.

request.AddUrlSegment will replace matching token in request. Here, in above example it will replace id with value 123 when it will find id in request.

With request.AddHeader you can easily add HTTP headers in your request.

And finally you can execute your request using client.Execute(request). You can use the response object to parse your data.

RestSharp also have async support.

```
            client.ExecuteAsync(request, response => {

     Console.WriteLine(response.Content);

 });
```

Using the Code

```
try

{

   var client = new RestClient();

   // This, of course, needs to be altered to match the

   // IP Address/Port of the server to which you are connecting

   client.BaseUrl = "http://192.128.65.42:80/";

   var request = new RestRequest();

   // The server's Rest method will probably return something

   // other than "duckbilledPlatypi" in your case

   request.Resource = "api/duckbilledPlatypi/";

   var response = client.Execute(request) as RestResponse;

   if (response != null && ((response.StatusCode == HttpStatusCode.OK) &&
     (response.ResponseStatus == ResponseStatus.Completed))) // It's probably not necessary to test
both

    {

        var arr = JsonConvert.DeserializeObject<JArray>(response.Content);

        foreach (JObject obj in arr)

        {

            var id = (string)obj["Id"];

            var platypusId = (double)obj["PlatypusId"]

            var platypusName = (string)obj["PlatypusName"];

            MessageBox.Show(string.Format("saw id of {0}, platypusId of {1},

              and platypusName of {2}", id, platypusId, platypusName));

        }

    }

    else if (response != null)

    {

        MessageBox.Show(string.Format

        ("Status code is {0} ({1}); response status is {2}",

               response.StatusCode, response.StatusDescription, response.ResponseStatus));

    }

}

catch (Exception ex)

{

   MessageBox.Show(ex.Message);

}

// Create a new RestClient and RestRequest

            var client = new RestClient("http://www.atavisticsoftware.com/");
            var request = new RestRequest ("demo/jsondbcount.php",Method.GET);
// ask for the response to be in JSON syntax

            request.RequestFormat = DataFormat.Json;

//send the request to the web service and store the response when it comes back

            var response = client.Execute (request);
```

```
// The next line of code will only run after the response has been received

// Create a new Deserializer to be able to parse the JSON object

            RestSharp.Deserializers.JsonDeserializer deserial= new JsonDeserializer();
//To deserialize into a simple variable, use the <Dictionary<string, string>> type


            var JSONObj = deserial.Deserialize<Dictionary<string, string>>(response);
            int rowCount =  JSONObj["Count"]; //rowCount will be 234 based on the example {"Count":234}
```

Authentication using RestSharp

```
RestClient client = new RestClient("http://www.newsblur.com");

            RestRequest login = new RestRequest("/api/login", Method.POST);
            login.AddParameter("username", "xxxxx");
            login.AddParameter("password", "xxxxx");

            IRestResponse response = client.Execute(login);
            CookieContainer cookiecon = new CookieContainer();

            if (response.StatusCode == HttpStatusCode.OK)
            {
                var cookie = response.Cookies.FirstOrDefault();
                cookiecon.Add(new Cookie(cookie.Name, cookie.Value, cookie.Path, cookie.Domain));
            }

            client.CookieContainer = cookiecon;


            RestRequest feeds = new RestRequest("/reader/feeds", Method.GET);

IRestResponse<Classes.FeedData> resp = client.Execute<Classes.FeedData>(feeds);
```

Restsharp to authenticate

```
//class definitions
        request = new RestRequest("http://allpoetry.com/login", Method.POST);
        request.AddParameter("utf8", "%E2%9C%93");
        request.AddParameter("authenticity_token", "Lr+JtLv7TkZrS73u5scRPbWhuUYDaVF7vd6lkKFF3FKYqNKHircT9v5WCT9EkneTJRsKXaA6nf6fiWopDB0INw==");
        request.AddParameter("referer", url);  // url to the page I want to go
        request.AddParameter("user[name]", "username");
        request.AddParameter("user[password]", "password");
        request.AddParameter("commit", "Log in");
        request.AddHeader("content-type", "application/x-www-form-urlencoded");
        response = client.Execute(request);
```

Popular Answer

```
request.Resource = url;
        request.Method = Method.GET;
        foreach (var cookie in response.Cookies)
        {
            request.AddCookie(cookie.Name , cookie.Value);  //this adds every cookie in the previous response.
```

```
         }
         response = client.Execute(request);
      //use the response as required
```

Parsing a JSON array using dynamic

```
using System;
using Newtonsoft.Json;

public class Program
{
	public static void Main()
	{
		var json = @"[
  {
    'email': 'john.doe@sendgrid.com',
    'timestamp': 1337197600,
    'smtp-id': '&lt;4FB4041F.6080505@sendgrid.com&gt;',
    'event': 'processed'
  },
  {
    'email': 'john.doe@sendgrid.com',
    'timestamp': 1337966815,
    'smtp-id': '&lt;4FBFC0DD.5040601@sendgrid.com&gt;',
    'category': 'newuser',
    'event': 'clicked'
  },
  {
    'email': 'john.doe@sendgrid.com',
    'timestamp': 1337969592,
    'smtp-id': '&lt;20120525181309.C1A9B40405B3@Example-Mac.local&gt;',
    'event': 'processed'
  }
  ]";

		dynamic stuff = JsonConvert.DeserializeObject(json);

		foreach (var s in stuff)
		{
		    Console.WriteLine(s.timestamp);

		}

	}
}
```

```
1337197600
1337966815
1337969592
```

```
----------------------------------------------------
string json = @"[
  {
    'Title': 'Json.NET is awesome!',
    'Author': {
      'Name': 'James Newton-King',
      'Twitter': '@JamesNK',
      'Picture': '/jamesnk.png'
    },
    'Date': '2013-01-23T19:30:00',
    'BodyHtml':
'&lt;h3&gt;Title!&lt;/h3&gt;\r\n&lt;p&gt;Content!&lt;/p&gt;'
  }
]";

dynamic blogPosts = JArray.Parse(json);

dynamic blogPost = blogPosts[0];

string title = blogPost.Title;

Console.WriteLine(title);
// Json.NET is awesome!

string author = blogPost.Author.Name;

Console.WriteLine(author);
// James Newton-King

DateTime postDate = blogPost.Date;

Console.WriteLine(postDate);
// 23/01/2013 7:30:00 p.m.
```

Parsing JSON Object using JObject

```
string json = @"{
  'channel': {
    'title': 'James Newton-King',
    'link': 'http://james.newtonking.com',
    'description': 'James Newton-King\'s blog.',
    'item': [
      {
        'title': 'Json.NET 1.3 + New license + Now on CodePlex',
        'description': 'Announcing the release of Json.NET 1.3, the MIT license
and the source on CodePlex',
        'link': 'http://james.newtonking.com/projects/json-net.aspx',
        'category': [
          'Json.NET',
          'CodePlex'
        ]
      },
      {
        'title': 'LINQ to JSON beta',
        'description': 'Announcing LINQ to JSON',
        'link': 'http://james.newtonking.com/projects/json-net.aspx',
        'category': [
          'Json.NET',
          'LINQ'
        ]
      }
    ]
  }
}";

JObject rss = JObject.Parse(json);

string rssTitle = (string)rss["channel"]["title"];

Console.WriteLine(rssTitle);
// James Newton-King

string itemTitle = (string)rss["channel"]["item"][0]["title"];

Console.WriteLine(itemTitle);
// Json.NET 1.3 + New license + Now on CodePlex

JArray categories = (JArray)rss["channel"]["item"][0]["category"];

Console.WriteLine(categories);
// [
//   "Json.NET",
//   "CodePlex"
// ]

string[] categoriesText = categories.Select(c => (string)c).ToArray();

Console.WriteLine(string.Join(", ", categoriesText));

// Json.NET, CodePlex
```

Querying JSON with LINQ

```
string json = @"{
  'channel': {
    'title': 'James Newton-King',
    'link': 'http://james.newtonking.com',
    'description': 'James Newton-King\'s blog.',
    'item': [
      {
        'title': 'Json.NET 1.3 + New license + Now on CodePlex',
        'description': 'Announcing the release of Json.NET 1.3,
the MIT license and the source on CodePlex',
        'link': 'http://james.newtonking.com/projects/json-
net.aspx',
        'category': [
          'Json.NET',
          'CodePlex'
        ]
      },
      {
        'title': 'LINQ to JSON beta',
        'description': 'Announcing LINQ to JSON',
        'link': 'http://james.newtonking.com/projects/json-
net.aspx',
        'category': [
          'Json.NET',
          'LINQ'
        ]
      }
    ]
  }
}";

JObject rss = JObject.Parse(json);

var postTitles =
    from p in rss["channel"]["item"]
    select (string)p["title"];

foreach (var item in postTitles)
{
    Console.WriteLine(item);
}
//LINQ to JSON beta
//Json.NET 1.3 + New license + Now on CodePlex

var categories =
    from c in
rss["channel"]["item"].Children()["category"].Values<string>()
    group c by c
    into g
    orderby g.Count() descending
    select new { Category = g.Key, Count = g.Count() };

foreach (var c in categories)
{
    Console.WriteLine(c.Category + " - Count: " + c.Count);
}
//Json.NET - Count: 2
//LINQ - Count: 1
//CodePlex - Count: 1
```

Read JSON from a file

```
JObject o1 =
JObject.Parse(File.ReadAllText(@"c:\videogames.json"));

// read JSON directly from a file
using (StreamReader file = File.OpenText(@"c:\videogames.json"))
using (JsonTextReader reader = new JsonTextReader(file))
{
    JObject o2 = (JObject)JToken.ReadFrom(reader);
}
```

Write JSON to a file

```
JObject videogameRatings = new JObject(
    new JProperty("Halo", 9),
    new JProperty("Starcraft", 9),
    new JProperty("Call of Duty", 7.5));

File.WriteAllText(@"c:\videogames.json",
videogameRatings.ToString());

// write JSON directly to a file
using (StreamWriter file =
File.CreateText(@"c:\videogames.json"))
using (JsonTextWriter writer = new JsonTextWriter(file))
{
    videogameRatings.WriteTo(writer);
```

Convert JSON to Collection

```
string json = @"{
  'd': [
    {
      'Name': 'John Smith'
    },
    {
      'Name': 'Mike Smith'
    }
  ]
}";

JObject o = JObject.Parse(json);

JArray a = (JArray)o["d"];

IList<Person> person = a.ToObject<IList<Person>>();

Console.WriteLine(person[0].Name);
// John Smith

Console.WriteLine(person[1].Name);
// Mike Smith
```

Using JsonConverter

```
static void Main(string[] args)
{
    stringjsonData = @ "{
    'FirstName': 'Jignesh', 'LastName': 'Trivedi'
}
";
var
myDetails = JsonConvert.DeserializeObject < MyDetail > (jsonData);
Console.WriteLine(string.Concat("Hi ", myDetails.FirstName, " " + myDetails.LastName));
Console.ReadLine();
    }
    public class MyDetail
    {
        public string FirstName {
            get;
            set;
        }
        public string LastName {
            get;
            set;
        }
    }
```

Querying JSON with SelectToken

SelectToken Example

```
JObject o = JObject.Parse(@"{
  'Stores': [
    'Lambton Quay',
    'Willis Street'
  ],
  'Manufacturers': [
    {
      'Name': 'Acme Co',
      'Products': [
        {
          'Name': 'Anvil',
          'Price': 50
        }
      ]
    },
    {
      'Name': 'Contoso',
      'Products': [
        {
          'Name': 'Elbow Grease',
          'Price': 99.95
        },
        {
          'Name': 'Headlight Fluid',
          'Price': 4
        }
      ]
    }
  ]
}");

string name = (string)o.SelectToken("Manufacturers[0].Name");
```

```
// Acme Co

decimal productPrice = (decimal)o.SelectToken("Manufacturers[0].Products[0].Price");
// 50

string productName = (string)o.SelectToken("Manufacturers[1].Products[0].Name");
// Elbow Grease
```

SelectToken with JSONPath

```
JObject o = JObject.Parse(@"{
  'Stores': [
    'Lambton Quay',
    'Willis Street'
  ],
  'Manufacturers': [
    {
      'Name': 'Acme Co',
      'Products': [
        {
          'Name': 'Anvil',
          'Price': 50
        }
      ]
    },
    {
      'Name': 'Contoso',
      'Products': [
        {
          'Name': 'Elbow Grease',
          'Price': 99.95
        },
        {
          'Name': 'Headlight Fluid',
          'Price': 4
        }
      ]
    }
  ]
}");

// manufacturer with the name 'Acme Co'
JToken acme = o.SelectToken("$.Manufacturers[?(@.Name == 'Acme Co')]");

Console.WriteLine(acme.ToString());
// { "Name": "Acme Co", Products: [{ "Name": "Anvil", "Price": 50 }] }

// name of all products priced 50 and above
IEnumerable<JToken> pricyProducts = o.SelectTokens("$..Products[?(@.Price >= 50)].Name");

foreach (JToken item in pricyProducts)
{
    Console.WriteLine(item.ToString());
}
```

```
// Anvil
// Elbow Grease
```

SelectToken with LINQ

```
IList<string> storeNames = o.SelectToken("Stores").Select(s =>
   (string)s).ToList();
// Lambton Quay
// Willis Street

IList<string> firstProductNames = o["Manufacturers"].Select(m =>
   (string)m.SelectToken("Products[1].Name")).ToList();
// null
// Headlight Fluid

decimal totalPrice = o["Manufacturers"].Sum(m =>
   (decimal)m.SelectToken("Products[0].Price"));
// 149.95
```

JSON Path Finder http://jsonpathfinder.com/

Serialize an Object

```
// {
//   "Email": "james@example.com",
//   "Active": true,
//   "CreatedDate": "2013-01-20T00:00:00Z",
//   "Roles": [
//     "User",
//     "Admin"
//   ]
// }

public class Account
{
    public string Email { get; set; }
    public bool Active { get; set; }
    public DateTime CreatedDate { get; set; }
    public IList<string> Roles { get; set; }
}

Account account = new Account
{
    Email = "james@example.com",
    Active = true,
    CreatedDate = new DateTime(2013, 1, 20, 0, 0, 0,
DateTimeKind.Utc),
    Roles = new List<string>
    {
        "User",
        "Admin"
    }
};

string json = JsonConvert.SerializeObject(account,
Formatting.Indented);

Console.WriteLine(json);
```

Compare JSON files

```
string filePath1 =@"D:\Test\";
string filePath2 =@"D:\Test1\";
string result =@"D:\Result\result.txt";
DirectoryInfo d = new DirectoryInfo(filePath1);
FileInfo[] Files1 = d.GetFiles("*.json"); //Getting JSON files
string[] Files2 = Directory.GetFiles(filePath2 ,"*. json ", SearchOption.TopDirectoryOnly);
Console.WriteLine("--- # in Path 1 : "+ Files1.Length);
Console.WriteLine("--- # in Path 2 : "+ Files2.Length);
// Write file using StreamWriter
using (StreamWriter writer = new StreamWriter(result))
{
  writer.WriteLine ("--- # in Path 1 : "+ Files1.Length);
  writer.WriteLine ("--- # in Path 2 : "+ Files2.Length);

foreach(FileInfo file in Files )
{
         if (!File.Exists (filePath2+ file.Name)) {
              Console.WriteLine("File doesn’t exists in filePath2...");
              writer.WriteLine ("File doesn’t exists in filePath2...");
            } else {
      var j1 = JToken.Parse(Files.ReadAllText(filePath1+ file.Name));
      var j2 = JToken.Parse(Files.ReadAllText(filePath2+ file.Name));

           var diff = JsonDifferentiator.Differentiate(j1,j2);

         if(diff!=null)
         {
           writer.WriteLine ("JSON File data Mismatch...");
           Console.WriteLine("JSON File data Mismatch…”);
         } else {
            writer.WriteLine ("JSON File data Match...");
           Console.WriteLine("JSON File data Match…”);
         }
}
}
 }
```