Executing multipart requests / Process multipart or complex responses

vutil provides a REST API to execute multipart requests. This API handles the multipart request and response as well. This API provides options to configure your multipart request and process the multipart response as well.

API Endpoint:

API Endpoint for executing multipart request/process response is

```
POST {{vutilBaseURL}}/request
```

Here

- {{vutilBaseURL}} is the variable, and the value is the base URL of the vutil server.
- and request body has the following structure

Request Body

```
{
        "method": "<STRING>",
        "url": "<STRING>",
        "headers": "<OBJECT>",
        "multipart": "<ARRAY<Part>>",
        "formData": "<FormDataObject>"
        "requestOptions": "<RequestOptionsObject>",
        "responseOptions": "<ResponseOptionsObject>"
}
```

here
 * method is the request method,
 * url is the request URL which will be executed,
 * headers is JSON object containing the request headers,
 * multipart is the array of part objects,
 * requestOptions is a JSON object to configure the API request and
 * responseOptions is a JSON object containing various options to process the API response part.

And each individual part structure is as follows:

Part

```
{
        "headers": "<OBJECT>",
        "body": "<STRING>" || "<PartBody>"
}
```

here
 * headers is JSON object containing the part headers and
 * body can be either string or of type PartBody.

The structure of PartBody is as follows:

PartBody

```
{
        "filePath": "<STRING>",
        "body": "<Array<Part>>" //Optional - for nested multipart body
}
```

here
 * filePath is the absolute file path on the machine where vutil is installed. vutil will automatically attach this file.
 * body is the array of part objects. It will be used for the nested multipart body.
The structure of FormDataObject is as follows:

FormDataObject

```
{
        "key1": "<SimpleValue>" || "<FileObject>" || "Array<FileObject>",
        ...
}
```

here
 * key1 is the name of the form data parameter and value can be one of SimpleValue or FileObject or array of file objects.
 * SimpleValue can be any type like string, number, boolean, JSON object etc.

The structure of FileObject is as follows:

FileObject

```
{
        "filePath": "<STRING>",
        "options": {
                "fileName": "<STRING>",
                "contentType": "<STRING>"
        }
}
```

here
 * filePath is the absolute file path of the local system where the vutil server is running.
 * fileName is the name of the file, by default file name will be extracted from the filePath parameter. This is an optional parameter.
 * contentType is the content type to be sent for this part. This is an optional parameter.

The structure of RequestOptionsObject is as follows:

RequestOptionsObject

```
{
        "preambleCRLF": "<BOOLEAN>",
        "postambleCRLF": "<BOOLEAN>",
        "timeout": "<NUMBER>",
        "oauth": "<OAuth1Object>"
}
```

here
 * preambleCRLF: append a newline/CRLF before the boundary of your `multipart/form-data` request.
 * postambleCRLF: append a newline/CRLF at the end of the boundary of your `multipart/form-data` request.
 * timeout: Integer containing the number of milliseconds to wait for a server to send response headers (and start the response body) before aborting the request.
 * oauth: Set this property only if you are using OAuth 1.0, just set the value as OAuth1Object. For other types of authorizations, please just set the Authorization property of the test case.

The structure of OAuth1Object is as follows:

OAuth1Object

```
{
        "consumer_key": "<STRING>", //consumer key
        "consumer_secret": "<STRING>", //consumer secret
        "token": "<STRING>", //Access token key
        "token_secret": "<STRING>" //Access token secret
}
```

The structure of ResponseOptionsObject is as follows:

ResponseOptionsObject

```
{
        "parse": "<BOOLEAN>",
        "process": "<ProcessObject>"
}
```

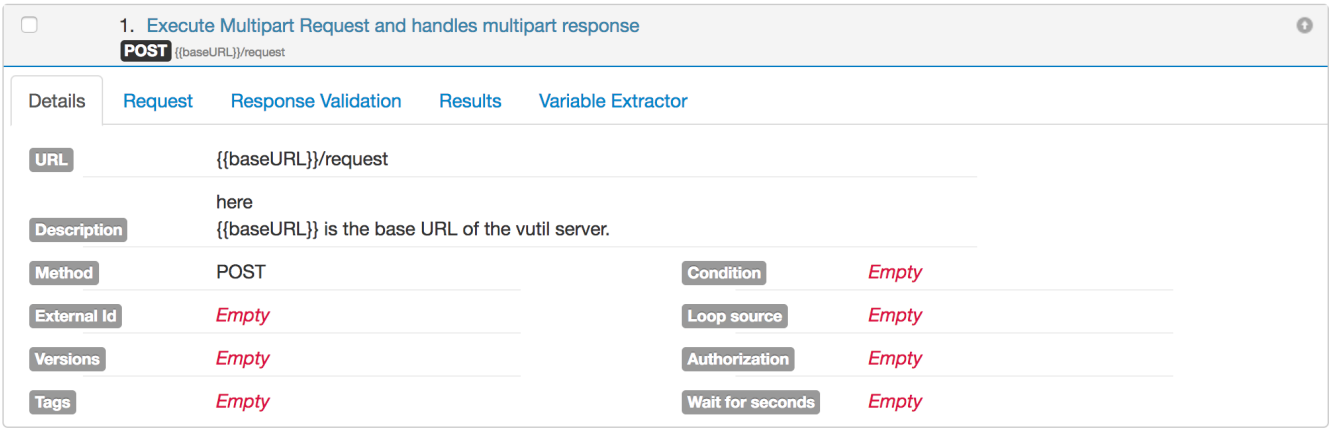
here
 * parse is the boolean flag, if true, the response will be parsed into parts otherwise response string will be returned directly and
 * process is a JSON object to process the part body according to the Content-Type of the part. In ProcessObject, the key is a substring of the response content type and value can be one of the following:

1. json: if you want to parse the JSON response part as a JSON object.
2. xml2json: if you want to convert the XML response part into a JSON object
3. csv2json: if you want to convert the CSV response part into a JSON object
4. xls2json: if you want to convert the .xlsx or .xls response part into JSON object
5. base64: if you want to encode the response part as base64 encoded string
6. blank: if you want to ignore the part response
7. checksum: if you want to calculate the checksum of the part response
8. string: if you want to return the part response as a string.

* You may also specify "default" as key and value will be the processor name.

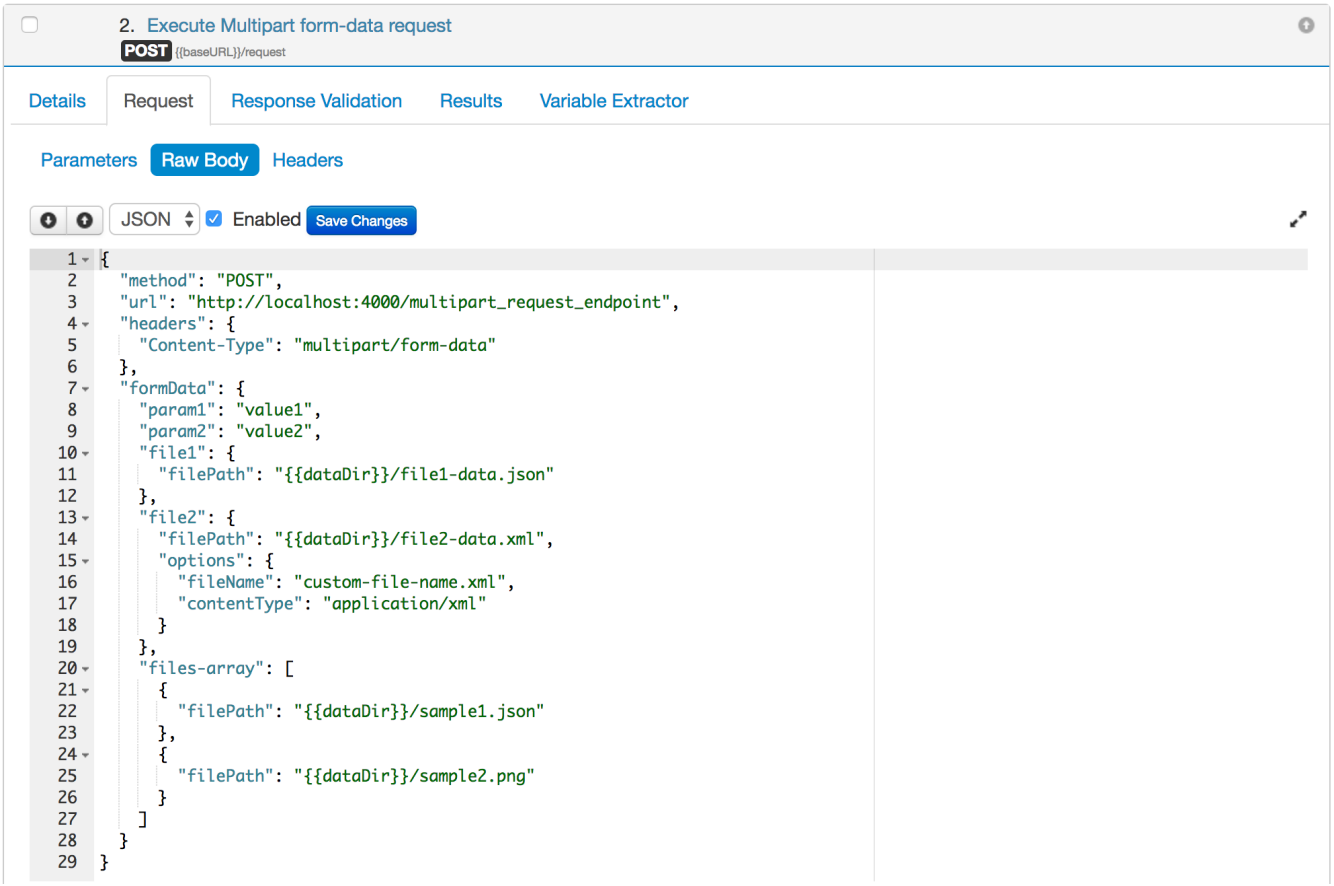
Let us take some examples on, how you can execute multipart requests with the help of vutil API. Then, later on, we will cover, how we can handle the multipart response.

First create a test case which will execute the request API of the vutil server. This is a POST API.



Now, we will set the multipart data to be sent in the request body of this test case. We will cover the multiple scenarios of setting multipart data to this API.

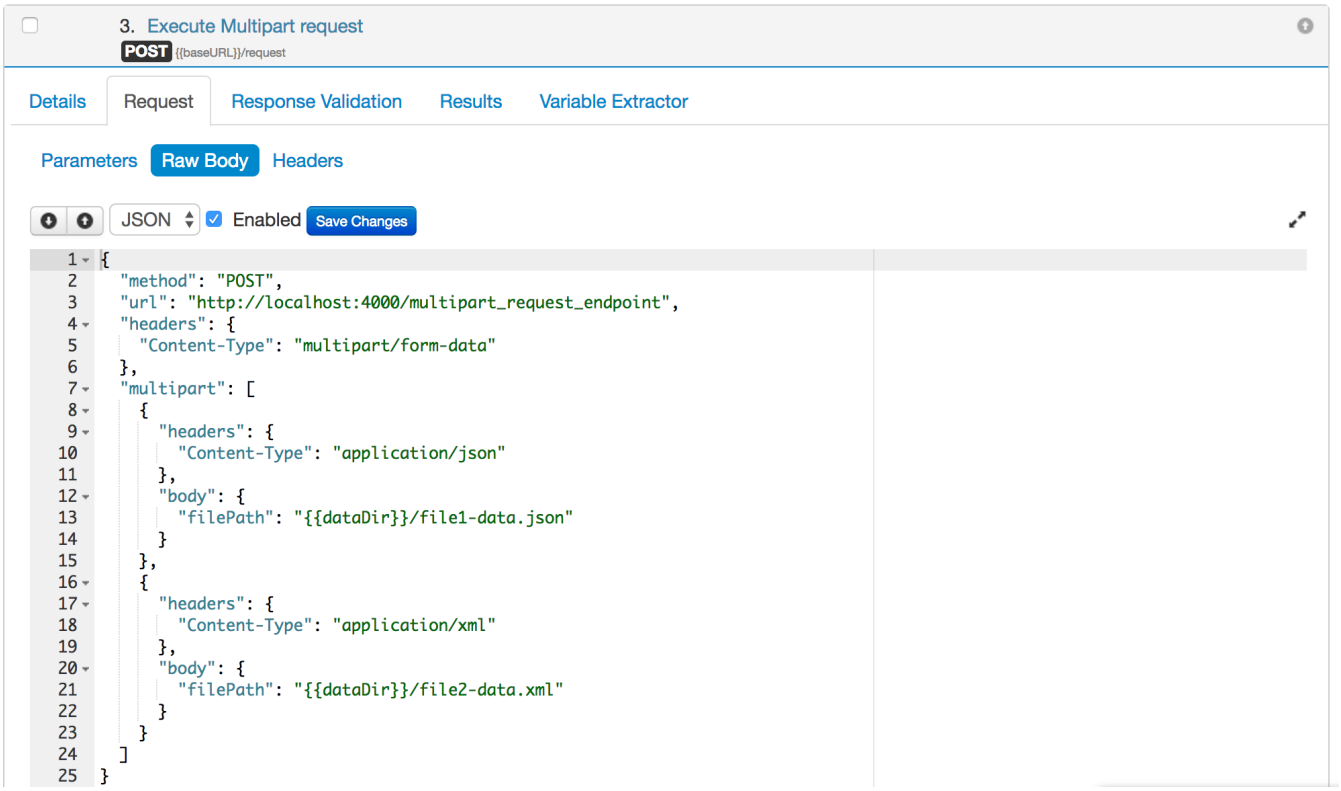
Example 1: Executing multipart/form-data request. Suppose I want to submit a form having some attached files, then I can configure the request body like this.



The actual request body of this POST request will look like this:

```
----------------------------967739397936203919086726
Content-Disposition: form-data; name="param1"

value1
----------------------------967739397936203919086726
Content-Disposition: form-data; name="param2"

value2
----------------------------967739397936203919086726
Content-Disposition: form-data; name="file1"; filename="file1-data.json"
Content-Type: application/json

Contents of file1-data.json
----------------------------967739397936203919086726
Content-Disposition: form-data; name="file2"; filename="custom-file-name.xml"
Content-Type: application/xml

Contents of file2-data.xml
----------------------------967739397936203919086726
Content-Disposition: form-data; name="files-array"; filename="sample1.json"
Content-Type: application/json

Contents of sample1.json
----------------------------967739397936203919086726
Content-Disposition: form-data; name="files-array"; filename="sample2.png"
Content-Type: application/png

Contents of sample2.png
----------------------------967739397936203919086726--
```

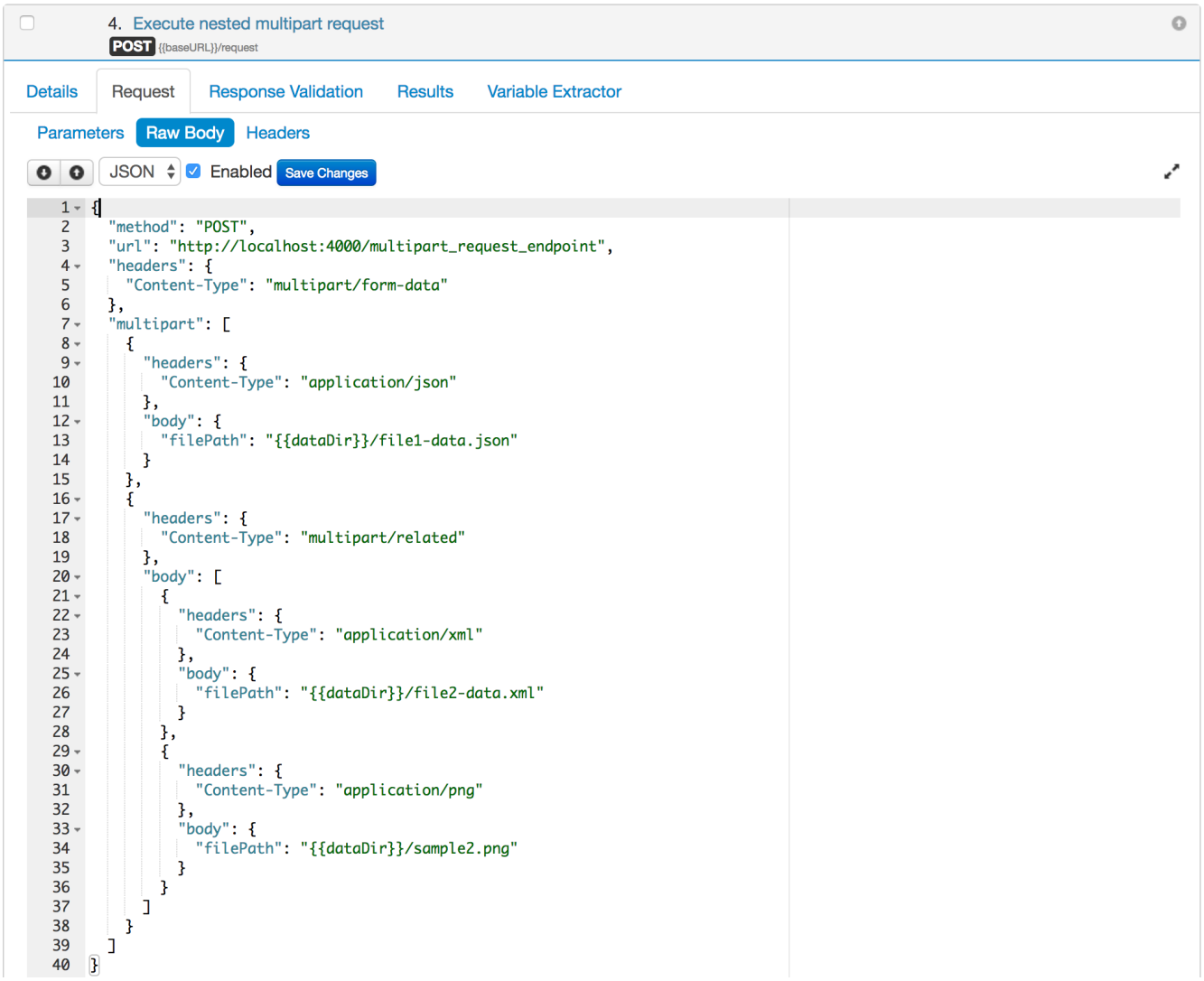
Example 2: Executing multipart request. Suppose I want to send multipart data, then I can configure the request body like this.



The actual request body for this POST request will look like this:

```
----------------------------967739397936203919086726
Content-Type: application/json

Contents of file1-data.json
----------------------------967739397936203919086726
Content-Type: application/xml

Contents of file2-data.xml
----------------------------967739397936203919086726--
```

Example 3: Executing nested multipart/related request. Suppose I want to send nested multipart body, then I can configure the request body like this.
In the below screenshot, we are sending two parts. The first part is a simple JSON file and the second part is again a multipart body. So we define the second part's body as an array of parts.




```
----------------------------967739397936203919086726
Content-Type: application/json

Contents of file1-data.json
----------------------------967739397936203919086726
Content-Type: multipart/related; boundary=6383052416266960721219 75;
transfer-encoding: chunked

----------------------------6383052416266960721219 75
Content-Type: application/xml

Contents of file2-data.xml
----------------------------6383052416266960721219 75
Content-Type: application/png

Contents of sample2.png
----------------------------6383052416266960721219 75--
----------------------------967739397936203919086726--
```

Handling of API response:

We will cover different examples of handling of API responses.

Example 1: Suppose API response is a normal JSON object.

If no responseOptions are set in the request body then vutil will automatically parse the JSON response and will provide the following response:

```
{
  "body": {
    "hello": "world" //Sample JSON Response
  },
  "headers": {
    "content-type": "application/json",
    "date": "Tue, 02 May 2017 19:16:27 GMT",
    "connection": "close"
  },
  "statusCode": 200
}
```

If you don't want to parse the API response then you may set the parse option to false in responseOptions object like below:

```
{
        "responseOptions": {
                "parse": false
        }
}
```

then the API response will be as follows:

```
{
  "body": "{\"hello\": \"world\"}", //JSON Response in string form
  "headers": {
    "content-type": "application/json",
    "date": "Tue, 02 May 2017 19:16:27 GMT",
    "connection": "close"
  },
  "statusCode": 200
}
```

Example 2: Suppose API response is a multipart response.

Now suppose, the API returns a multipart response. The response has three parts, one part contains JSON response, the second part contains XML response and the third part contains the PNG file. Now, how you will validate such response in vREST.

To process such type of API response, you may need to process the API response first before it is read by vREST. Simply set the responseOptions like below:

```
{
        "responseOptions": {
                "parse": true,
                "process": {
                        "json": "json", //json processor simply parses the JSON response
                        "xml": "xml2json", //xml processor converts xml into JSON
                        "png": "checksum" //checksum processor calculates the md5 checksum of the part data
                }
        }
}
```

And the API response might look like:

```
{
  "body": {
    "parts": [
      { //first part response
        "body": {
                        "hello": "world"
                },
        "bodylen": 20,
        "headers": {
          "content-type": "application/json"
        }
      },
```

```
      { //second part response
        "body": {
          "iterations": {
            "iteration": {
              "request": {
                "body": {
                  "name": {},
                  "email": {
                    "#text": "john.doe@example.com"
                  },
                  "designation": {
                    "#text": "Chief Technical Officer"
                  },
                  "organization": {
                    "#text": "example.com"
                  },
                  "country": {
                    "#text": "India"
                  },
                  "aboutMe": {
                    "#text": "Please write something about me…"
                  },
                  "twitterId": {
                    "#text": "fake.john.doe"
                  },
                  "facebookId": {
                    "#text": "fake.john.doe"
                  },
                  "githubId": {
                    "#text": "fake.john.doe"
                  },
                  "createdOn": {
                    "#text": "2014-05-03T06:28:45.479Z"
                  }
                }
              }
            }
          }
        },
        "bodylen": 630,
        "headers": {
          "content-disposition": "form-data; name=\"file2\"; filename=\"small.xml\"",
          "content-type": "application/xml"
        }
      },
      { //third part response
              "body": "63c3998d81a7bdbac448b4c76d4d08f3", //md5 checksum of the png file
        "bodylen": 97955,
        "headers": {
                "content-type": "application/png"
        }
      }
    ]
  },
  "headers": { //response headers
    "content-type": "multipart/form-data; boundary=--------------------------566038963922264557922999",
    "date": "Tue, 02 May 2017 12:27:11 GMT",
    "connection": "close",
    "transfer-encoding": "chunked"
  },
  "statusCode": 200 //API response status code
}
```

Example 3: Suppose API response is a spreadsheet file (xlsx / xls).

Now suppose, the API returns a spreadsheet file as a response. To process such type of API response, you may need to process the API response first before it is read by vREST. Simply set the responseOptions like below:

```
{
        ...
        "responseOptions": {
                "parse": true,
                "process": {
                        "default": "xls2json", //xls2json processor converts xlsx or xls into JSON
                }
        }
}
```

And the API response might look like:

```
{
  "body": [
        {
          "key": "key1",
          "value": "value1"
         },
    {
          "key": "key2",
          "value": "value2"
        }
  ],
  "headers": { //response headers
    "content-type": "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
    "date": "Fri, 17 May 2019 11:32:29 GMT",
    "connection": "close"
  },
  "statusCode": 200 //API response status code
}
```