Problem statement

When you make a POST request to an API, you have to encode the data that forms the body of the request in some way.

HTML forms provide three methods of encoding –

* application/x-www-form-urlencoded (default)
* multipart/form-data
* text/plain

The specifics of the formats don’t matter to most developers. The important point to be considered is, when you are writing client-side code, use multipart/form-data when your form includes any **<input type=”file”>** elements. This means that using multipart/form-data, a request can contain a raw file along with other text components.

In Apigee, there is a seamless support for parsing and creating POST requests with encoding form-urlencoded and text/plain (which can include json or xml content depending upon the Content-Type header).

But, when it comes to creating or parsing multipart/form-data requests, it is often the case that you end up writing custom code. There are two cases which can be associated with multipart/form-data requests –

* **Case 1**: The back-end expects a multipart/form-data request and you need to create such a request from Apigee
* **Case 2**: Any client which sends a request to Apigee, can send a multipart/form-data request and you need to parse the request in order to extract or manipulate individual text fields/files

Both problems and their solutions have been discussed below.

The Java code can be customized as per your requirement.

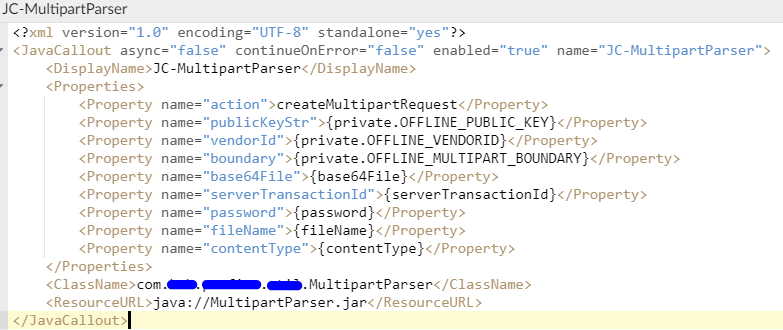
If you need help in sending multipart/form-data requests from POSTMAN, refer this post - <https://stackoverflow.com/questions/16015548/tool-for-sending-multipart-form-data-request>

Solution: Case 1

In our use case, the client made an application/json **POST** request to Apigee containing Base64 encoded string of the actual pdf file along with other string fields.

The back-end which Apigee connected to expected a multipart/form-data request with the raw binary pdf file and remaining text fields, which meant that the json request from client had to be transformed into multipart/form-data which had one file input containing the raw pdf and other text inputs.

**Mime4J** library was used for multipart/form-data creation.



*Sample Apigee configuration for Java callout*

In the Java code, take a look at MultipartParser.java class and ‘**createMultipartRequest**’ section –

**if** (*action*.equals(*constants*.CREATE\_METHOD\_NAME))

Every multipart request, requires a *boundary,* which separates each component (file or text) within the body. You can create this boundary dynamically or keep a constant boundary which is what we did.

Since, the request from client was application/json, we could extract each field separately from JSON in Apigee and send it to the Java code which stored these values in **String** variables.

The next challenge was converting the Base64 file into a raw pdf. This was a bit tricky considering that a raw document is binary in nature, which meant once the conversion was completed, we **could not** deal with string and rather had to deal in streams.

The final output was a multipart/form-data request, sent as a Java InputStream to Apigee which Apigee then sent to the back-end.

Note – If there is a requirement to create multipart/form-data request without a raw file, you need not write a Java code. The following article can be helpful to fulfil that requirement using JavaScript.

<https://community.apigee.com/questions/25630/need-to-send-a-request-to-a-service-with-contentty.html>

Solution: Case 2

In our use case, the client made a multipart/form-data request which contained a raw file along with other text components. We needed to perform a few transformations on the text components without hampering the binary raw file and return back the updated multipart/form-data request

**Mime4J** library was used for multipart/form-data parsing and creation.



*Sample Apigee configuration for Java callout*

In the Java code, take a look at MultipartParser.java class and ‘**parseMultipartRequest**’ section –

**if** (*action*.equals(*constants*.PARSE\_METHOD\_NAME))

Since the client already made a multipart/form-data request, the complete request was available in Apigee’s **request** object.

In the Java code, we could access the request object directly from Apigee using Apigee jars (message-flow and expressions). The request contained a binary raw file and hence, we made sure not to deal with **String** as it could damage the file. Instead, we used mime type to check if the part was text/plain and only then dealt with String. In all other cases, we used **Streams.**

Once all text manipulations were completed, we created our own multipart and sent that as an InputStream back to Apigee which Apigee then sent to the back-end.

We also set debug variables wherever applicable in the Java code for easy debugging.

Note – In case you are not dealing with files in your multipart request and want to extract or perform simple text manipulations, then you can refer the following article and use JavaScript instead –

<https://community.apigee.com/questions/36743/how-to-extract-multipartform-data-from-post-reques.html>