In **HTTP** there are **two ways to POST data**:

**application/x-www-form-urlencoded** and

**multipart/form-data**.

**Regarding Restful API over HTTP** the most popular content-types I came in touch with are **application/xml** and **application/json**.

I understand that most browsers are only able to upload files if multipart/form-data is used. Is there any additional guidance when to use one of the encoding types in an API context (no browser involved)? This might e.g. be based on:

* data size
* existence of non-ASCII characters
* existence on (unencoded) binary data
* the need to transfer additional data (like filename)

Summary; if you have binary (non-alphanumeric) data (or a significantly sized payload) to transmit, use multipart/form-data. Otherwise, use application/x-www-form-urlencoded.

The MIME types you mention are the two Content-Type headers for HTTP POST requests that user-agents (browsers) must support.

The purpose of both of those types of requests is to send a list of name/value pairs to the server. Depending on the type and amount of data being transmitted, one of the methods will be more efficient than the other. To understand why, you have to look at what each is doing under the covers.

For application/x-www-form-urlencoded, the body of the HTTP message sent to the server is essentially one giant query string -- name/value pairs are separated by the ampersand (&), and names are separated from values by the equals symbol (=). An example of this would be:

MyVariableOne=ValueOne&MyVariableTwo=ValueTwo

According to the [specification](http://www.w3.org/TR/html401/interact/forms.html):

[Reserved and] non-alphanumeric characters are replaced by `%HH', a percent sign and two hexadecimal digits representing the ASCII code of the character

That means that for each non-alphanumeric byte that exists in one of our values, it's going to take three bytes to represent it. For large binary files, tripling the payload is going to be highly inefficient.

That's where multipart/form-data comes in. With this method of transmitting name/value pairs, each pair is represented as a "part" in a MIME message (as described by other answers). Parts are separated by a particular string boundary (chosen specifically so that this boundary string does not occur in any of the "value" payloads). Each part has its own set of MIME headers like Content-Type, and particularly Content-Disposition, which can give each part its "name." The value piece of each name/value pair is the payload of each part of the MIME message. The MIME spec gives us more options when representing the value payload -- we can choose a more efficient encoding of binary data to save bandwidth (e.g. base 64 or even raw binary).

Why not use multipart/form-data all the time? For short alphanumeric values (like most web forms), the overhead of adding all of the MIME headers is going to significantly outweigh any savings from more efficient binary encoding.

The content type "application/x-www-form-urlencoded" is inefficient for sending large quantities of binary data or text containing non-ASCII characters. The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data.

I don't think HTTP is limited to POST in multipart or x-www-form-urlencoded. The [Content-Type Header](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17) is orthogonal to the HTTP POST method (you can fill MIME type which suits you). This is also the case for typical **HTML representation based webapps** (e.g. json payload became very popular for transmitting payload for ajax requests).

**Regarding Restful API over HTTP** the most popular content-types I came in touch with are **application/xml** and **application/json**.

**application/xml:**

* data-size: XML very verbose, but usually not an issue when using compression and thinking that the write access case (e.g. through POST or PUT) is much more rare as read-access (in many cases it is <3% of all traffic). Rarely there where cases where I had to optimize the write performance
* existence of non-ascii chars: you can use utf-8 as encoding in XML
* existence of binary data: would need to use base64 encoding
* filename data: you can encapsulate this inside field in XML

**application/json**

* data-size: more compact less that XML, still text, but you can compress
* non-ascii chars: json is utf-8
* binary data: base64 (also see [json-binary-question](https://stackoverflow.com/questions/1443158/binary-data-in-json-string-something-better-than-base64))
* filename data: encapsulate as own field-section inside json

--HereGoes

Content-Disposition: form-data; name="myJsonString"

Content-Type: application/json

{"foo": "bar"}

--HereGoes

Content-Disposition: form-data; name="photo"

Content-Type: image/jpeg

Content-Transfer-Encoding: base64

<...JPEG content in base64...>

--HereGoes--

What is the difference between

request.ContentType = "application/json; charset=utf-8";

and

webRequest.ContentType = "application/x-www-form-urlencoded";

The first case is telling the web server that you are posting JSON data as in:

{ Name : 'John Smith', Age: 23}

The second option is telling the web server that you will be **encoding** the parameters in the URL as in:

Name=John+Smith&Age=23

webRequest.ContentType = "application/x-www-form-urlencoded";

1. Where does **application/x-www-form-urlencoded**'s name come from?

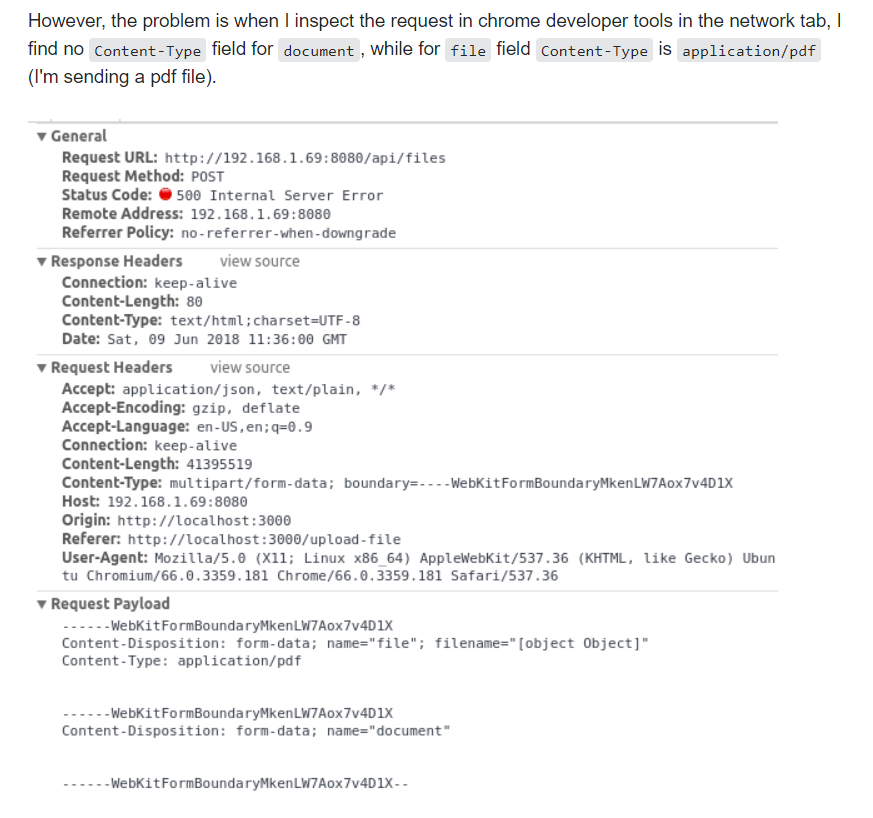
If you send HTTP **GET** request, you can use query parameters as follows:

http://example.com/path/to/page**?name=ferret&color=purple**

The content of the fields is encoded as a query string. The application/x-www-form- urlencoded's name come from the previous url query parameter **but the query parameters is in where the body of request instead of url**.

The whole form data is sent as a long query string.The query string contains **name- value** pairs separated by **&** character

e.g. field1=value1&field2=value2



**This is an example full HTTP request:**

Again, **the overall HTTP content type** (**multipart/form-data; boundary="----=\_Part\_4\_35071402.1568998611352"**) is automatically generated. The boundary value is random and you cannot control that (you do not need to). Message ID and MIME version are automatically added as part of the HTTP request

**POST** /api/files.upload?channels=THNMAZN8G HTTP/1.1

**Content-Type**: **multipart/form-data**; boundary="----=\_Part\_4\_35071402.1568998611352"

User-Agent: Boomi Http Transport

Message-ID: <519117923.5.1568998611383.JavaMail.W10CV0B0X2$@W10CV0B0X2>

MIME-Version: 1.0

Connection: close

**Authorization**: Bearer xoxp-75624234-834346893544234-1256634555-876ff8fc520e18c5fdd7b30ac5f8988e

Cache-Control: no-cache

Pragma: no-cache

Host: [slack.com](http://slack.com/)

Accept: text/html, image/gif, image/jpeg, \*; q=.2, \*/\*; q=.2

Content-Length: 214529

------=\_Part\_4\_35071402.1568998611352

Content-Type: application/octet-stream

Content-Transfer-Encoding: binary

Content-Disposition**: form-data; name="file"; filename=test.pdf**

**<<PDF\_Data>>**

------=\_Part\_4\_35071402.1568998611352—

# Sending a file and other form data together

If you need to send a file and other form data, you need to add multiple JSON objects. Combining the examples above,

'{

      "ID" : {

         "value" : "XYZ123ABC",

         "Content-Type" : "application/text-plain",

         "Content-Transfer-Encoding":"8bit",

         "type" : "data"

      },

"service-name" : {

         "value" : "Bob",

         "Content-Type" : "application/text-plain",

         "Content-Transfer-Encoding":"8bit",

         "type" : "data"

      },

  "file" : {

         "value" : "test.pdf",

         "Content-Type" : "application/octet-stream",

"Content-Transfer-Encoding":"binary",

         "type" : "key"

      }

}'

**post the contents of a file that contains JSON. The contents of the file look like this**:

{

"id”:99999999,

"orders":[

{

"ID”:8383838383,

"amount":0,

"slotID":36972026

},

{

"ID”:2929292929,

"amount":0,

"slotID":36972026

},

{

"ID”:4747474747,

"amount":0,

"slotID":36972026

}]

}

import requests

url = 'https://api.example.com/api/dir/v1/accounts/9999999/orders'

headers = {'Authorization' : ‘(some auth code)’, 'Accept' : 'application/json', 'Content-Type' : 'application/json'}

r = **requests.post**(url, data=open('example.json', 'rb'), headers=headers)

#!/usr/bin/env python3

import requests

import json

files = {'file': open(‘example.json’, 'rb')}

headers = {'Authorization' : ‘(some auth code)’, 'Accept' : 'application/json', 'Content-Type' : 'application/json'}

r = **requests.post**('https://api.example.com/api/dir/v1/accounts/9999999/orders', files=files, headers=headers)

import requests

import json

**with open('example.json') as json\_file:**

**json\_data = json.load(json\_file)**

**headers = {'Authorization' : ‘(some auth code)’, 'Accept' : 'application/json', 'Content-Type' : 'application/json'}**

**r = requests.post('https://api.example.com/api/dir/v1/accounts/9999999/orders', data=json.dumps(json\_data), headers=headers)**

with open('example.json') as json\_file:

json\_data = json.load(json\_file)

auth=('token', 'example')

r = **requests.post**('https://api.example.com/api/dir/v1/accounts/9999999/orders', **json=json\_data**, auth=auth)

**If you want to send a smaller file, send it as a string.**

contents = open('example.json', 'rb').read()

r = requests.post(url, data=contents, headers=headers)

We are using API Gateway in our project and are running into a weird issue. We have a virtualized REST API allowing us to send images to a server.

If I send my request directly to the server (through the backend URL), it handles it with success and saves the image as expected. But when I send the request to the virtualised API (through the frontend API), the gateway seems unable to understand the request. The requests are exactly the same, only the urls I use diverge.

The request is as follows:

POST /v1/img HTTP/1.1

Host: xx.xx.xx.xx:8065

User-Agent: curl/7.47.0

Accept: application/json

Cache-Control: no-cache

Content-Length: 15469

Expect: 100-continue

content-type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW; boundary=------------------------5180b2a73f429dfd

--------------------------5180b2a73f429dfd

Content-Disposition: form-data; name="image"; filename="testimage.jpg"

Content-Type: image/jpeg

(((((((((((((((((((((((((((((((((((((((((((((((((((ï¿½ï¿½ï¿½ï¿½"ï¿½ï¿½ï¿½ï¿½ï¿½ï¿½ï¿½@AJ(ï¿½ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½Ë†ï¿½3

iÊ»@(ï¿½rï¿½QQlU8bï¿½ï¿½5ï¿½ï¿½ï¿½qÄ�ï¿½uï¿½ï¿½2ï¿½R\*ï¿½Rï¿½q g,AYï¿½Wï¿½ï¿½Jï¿½Zï¿½[5+ï¿½ï¿½Qlï¿½JÛ¸ï¿½[ï¿½ï¿½Ò”ï¿½ï¿½ï¿½?ï¿½|&È³Lï¿½ï¿½.eï¿½\*8W5Â¹ï¿½ï¿½pï¿½kï¿½s\+ï¿½t>eï¿½PÔ›Ð®|

.......

--------------------------5180b2a73f429dfd

Content-Disposition: form-data; name="id"

testFrontend

--------------------------5180b2a73f429dfd--

**The Accept header should be changed to application/json in order to get a JSON response**

#

# [upload a file and post JSON data in the same request](https://forums.servicestack.net/t/how-to-upload-a-file-and-post-json-data-in-the-same-request/3318)

POST [http://ws-local.myhost.com/UploadFile?format=json 75](http://ws-local.myhost.com/UploadFile?format=json) HTTP/1.1  
Accept: multipart/form-data  
Authorization: IRA-HMAC

Content-Type: multipart/form-data; boundary=---------------------------–b4537661-9989-40c1-977b-41dd400fe6c9  
Content-Type: multipart/form-data  
Host: [ws-local.myhost.com 1](http://ws-local.myhost.com/)  
Content-Length: 491789  
Expect: 100-continue

–b4537661-9989-40c1-977b-41dd400fe6c9  
Content-Type: application/json; charset=utf-8  
Content-Disposition: form-data

{“File”:{“AccountId”:“6d1c818a-7b3a-40e4-ab72-cadf4eae7c85”}}  
–b4537661-9989-40c1-977b-41dd400fe6c9  
Content-Disposition: form-data; name=file; filename=MyFile.pdf; filename\*=utf-8’'MyFile.pdf

%PDF-1.7  
%  
455 0 obj  
…

Which parameter between (data / json / files) should be used,it's actually depends on a request header named ContentType(usually check this through developer tools of your browser),

when the Content-Type is application/x-www-form-urlencoded, code should be:

**requests.post(url, data=jsonObj)**

when the Content-Type is application/json, your code is supposed to be one of below:

**requests.post(url, json=jsonObj)**

**requests.post(url, data=jsonstr, headers={"Content-Type":"application/json"})**

when the Content-Type is multipart/form-data, it's used to upload files, so your code should be:

**requests.post(url, files=xxxx)**

In an application I am developing RESTful API and we want the client to send data as JSON. Part of this application requires the client to upload a file (usually an image) as well as information about the image.

I'm having a hard time tracking down how this happens in a single request. Is it possible to Base64 the file data into a JSON string? Am I going to need to perform 2 posts to the server? Should I not be using JSON for this?

**You can send the file and data over in one request using the**[**multipart/form-data**](http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2)**content type:**

"multipart/form-data" contains a series of parts. Each part is expected to contain a content-disposition header [RFC 2183] where the disposition type is "form-data", and where the disposition contains an (additional) parameter of "name", where the value of that parameter is the original field name in the form. For example, a part might contain a header:

Content-Disposition: form-data; name="user"

with the value corresponding to the entry of the "user" field.

You basically have three choices:

1. Base64 encode the file, at the expense of increasing the data size by around 33%, and add processing overhead in both the server and the client for encoding/decoding.
2. Send the file first in a multipart/form-data POST, and return an ID to the client. The client then sends the metadata with the ID, and the server re-associates the file and the metadata.
3. Send the metadata first, and return an ID to the client. The client then sends the file with the ID, and the server re-associates the file and the metadata.

**We are developing server with REST API, which accepts and responses with JSON.** The problem is, if you need to upload images from client to server.

Note: and also I am talking about a use-case where the entity (user) can have multiple files (carPhoto, licensePhoto) and also have other properties (name, email...), but when you create new user, you don't send these images, they are added after the registration process.

**1. Use multipart/form-data instead of JSON**

*good* : POST and PUT requests are as RESTful as possible, they can contain text inputs together with file.

*cons* : It is not JSON anymore, which is much easier to test, debug etc. compare to multipart/form-data

**2. Allow to update separate files**

**POST** request for creating new user does not allow to add images (which is ok in our use-case how I said at beginning), uploading pictures is done by **PUT** request as multipart/form-data to for example /users/4/carPhoto

*good* : Everything (except the file uploading itself) remains in JSON, it is easy to test and debug (you can log complete JSON requests without being afraid of their length)

*cons* : It is not intuitive, you cant POST or PUT all variables of entity **at once** and also this address /users/4/carPhoto can be considered more as a collection (standard use-case for REST API looks like this /users/4/shipments). Usually you cant (and dont want to) GET/PUT each variable of entity, for example users/4/name . You can get name with GET and change it with PUT at users/4. If there is something after the id, it is usually another collection, like users/4/reviews

**3. Use Base64**

Send it as JSON but encode files with Base64.

*good* : Same as first solution, it is as RESTful service as possible.

*cons* : Once again, testing and debugging is a lot worse (the body can have megabytes of data), there is increase in size and also in processing time in both - client and server

1. POST to /users to create the user entity.
2. POST the image to /images, making sure to return a Location header to where the image can be retrieved per the HTTP spec.
3. PATCH to /users/carPhoto and assign it the ID of the photo given in the Location header of step 2.

There's no easy solution. Each way has their pros and cons . But the canonical way is using the first option: multipart/form-data. As [W3 recommendation guide](http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2) says

The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data.

We aren't sending forms,really, but the implicit principle still applies. Using base64 as a binary representation, is incorrect because you're using the incorrect tool for accomplish your goal, in other hand, the second option forces your API clients to do more job in order to consume your API service. You should do the hard work in the server side in order to supply an easy-to-consume API. The first option is not easy to debug, but when you do it, it probably never changes.

Using multipart/form-data you're sticked with the REST/http philosophy. You can view an answer to similar question [here](https://stackoverflow.com/questions/4083702/posting-a-file-and-data-to-restful-webservice-as-json).

Another option if mixing the alternatives, you can use multipart/form-data but instead of send every value separate, you can send a value named payload with the json payload inside it. (I tried this approach using ASP.NET WebAPI 2 and works fine).

The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data

Is it possible to send binary data in JSON ?

If you need a JSON or XML wrapper to your file you'll have to text-encode it, Base 64 for example

The easiest way is to convert the binary data into string by using Base64 encoding. However this has space overhead as Base64 encoding means around +33% more memory.

convert the pdf document to base64, set that in a property of your json and on the server side take it out from base64 again to the binary and process further.

Today's web applications heavily rely on json for client server communication. Because json is a totally text based standard it goes very well until we need to send anything which is not text, like pdf files and images. And it is not a good idea that you implement a separate module to transfer binary files, it will defeat the whole purpose of using json.  
I didn't found a straightforward and simple tutorial to handle binary files with json.

As json only supports text so we have to convert binary file (pdf, image etc) in to a string. And then we can easily add it to a json field.

Hi Ivan, Definitely you can use it in your application. For example if you want to show a list of items in the browser, you can get a list of json objects from backend containing the information about the individual item of the list. Now if you want to show thumbnails along with their information, you can simply include the thumbnail image in the json object as a field. It will make it very easy to handle the code. And also you are not dependent on the image url to show images on the browser because you have included the image in the json itself, so you are free from creating and maintain images on the disk(or any web service) to create urls, you can directly get images from database (or any other storage) and send them directly to the browser. There are as many benefits as you can think of. But having said that, you must take in to account, the size of payload. You can't just include any size of binary content in json, because it will increase the size of json response, so you must consider the memory requirement of your project. But in general purpose scenarios it is good to use. Even I have created a framework to handle all this work in one of my projects, and out of there I have created this tutorial. I hope it answers your question.

Previous chapters talked about [regular POST](/http/http-post) and [multipart formpost](/http/http-multipart), and in your typical command lines you do them with -d or -F.

When do you use which of them?

As described in the chapters mentioned above, both these options send the specified data to the server. The difference is in how the data is formatted over the wire. Most of the time, the receiving end is written to expect a specific format and it expects that the sender formats and sends the data correctly. A client cannot just pick a format of its own choice.

Also, looks like your setting for **Single File Upload: Multipart Content-Type** is not set correctly. This field is for the MIME type of the file you’re trying to post, so when it gets formatted by the snap into a multipart request, it knows how to describe the file. So using “multipart/form-data” like you have set in the screenshot is not going to work.

**It looks like you’re posting an xlsx file. So in your case you can either use the specific mime type for the file “application/vnd.openxmlformats-officedocument.spreadsheetml.sheet” or preferably the generic binary stream content type “application/octet-stream”. By doing that you let the target server know what kind of file you’re posting.**

Try making those changes and see what your results are.

# [How to send a PDF file via JSON from my REST Service to my clients](https://stackoverflow.com/questions/10759970/how-to-send-a-pdf-file-via-json-from-my-rest-service-to-my-clients)

If you need a JSON or XML wrapper to your file you'll have to text-encode it, Base 64 for example

The easiest way is to convert the binary data into string by using Base64 encoding. In C#, this would mean a call to Convert.FromBase64String. However this has space overhead as Base64 encoding means around +33% more memory.

If you can get away with it, this is the least complicated solution. in case additional size is an issue you can think about zipping it up.