Ruby: How to post a file via HTTP as multipart/form-data?

I want to do an HTTP POST that looks like an HMTL form posted from a browser. Specifically, post some text fields and a file field.

Posting text fields is straightforward, there's an example right there in the net/http rdocs, but I can't figure out how to post a file along with it.

Net::HTTP doesn't look like the best idea. curb is looking good.

ruby | http | post

edited **Dec 4 '09 at 21:07**

asked **Oct 8 '08 at 18:31**

kch
15.2k 14 62 99
70% accept rate

feedback

protected by Flexo Jan 15 at 17:09

This question is protected to prevent "thanks!", "me too!", or spam answers by new users. To answer it, you must have earned at least 10 reputation on this site.

9 Answers

I like RestClient. It encapsulates net/http with cool features like multipart form data:

```
require 'rest_client'
RestClient.post('http://localhost:3000/foo',
  :name_of_file_param => File.new('/path/to/file'))
```

It also supports streaming.

`gem install rest-client` will get you started.

edited **Mar 20 '11 at 3:13**

eric
623 3 13

answered **Nov 25 '08 at 4:03**

Pedro
560 1 4 10

1 this looks purty – kch Nov 25 '08 at 4:12

I take that back, file uploads now work. Problem I'm having now is the server gives a 302 and the rest-client follows the RFC (which no browser does) and throws an exception (since browsers are supposed to warn about this behavior). THe other alternative is curb but I've never had any luck installing curb in windows. – Matt Wolfe Mar 6 '10 at 9:19

2 The API has changed a little since this was first posted, multipart now is invoked like: RestClient.post 'localhost:3000/foo';, :upload => File.new('/path/tofile')) See github.com/archiloque/rest-client for more details. – Clinton Mar 14 '10 at 9:16

This worked fine for me for posting to an Apache+Php script. – Matt Connolly May 20 '11 at 5:25

feedback

I can't say enough good things about Nick Sieger's multipart-post library.

It adds support for multipart posting directly to Net::HTTP, removing your need to manually worry about boundaries or big libraries that may have different goals than your own.

Here is a little example on how to use it from the README:

```
require 'net/http/post/multipart'

url = URI.parse('http://www.example.com/upload')
File.open("./image.jpg") do |jpg|
  req = Net::HTTP::Post::Multipart.new url.path,
    "file" => UploadIO.new(jpg, "image/jpeg", "image.jpg")
  res = Net::HTTP.start(url.host, url.port) do |http|
    http.request(req)
  end
end
```

You can check out the library here: http://github.com/nicksieger/multipart-post

or install it with:

```
$ sudo gem install multipart-post
```

edited **May 9 '10 at 7:42**

answered **Apr 8 '10 at 22:12**

eric

623 3 13

1 That one did it for me, exactly what I was looking for and exactly what should be included without the need for a gem. Ruby is so far ahead, yet so far behind. – Trey Apr 27 '10 at 0:45

awesome, this comes as a God send! used this to monkeypatch the OAuth gem to support file uploads. took me only 5 minutes. – Matthias Mar 9 '11 at 15:20

@matthias I'm trying to upload photo with OAuth gem, but failed. could you give me some example of your monkeypatch? – Hooopo May 16 '11 at 8:01

1 The patch was quite specific to my script (quick-and-dirty), but have a look at it and maybe you can some up with a more generic approach (gist.github.com/974084) – Matthias May 16 '11 at 8:19

feedback

`curb` looks like a great solution, but in case it doesn't meet your needs, you *can* do it with `Net::HTTP`. A multipart form post is just a carefully-formatted string with some extra headers. It seems like every Ruby programmer who needs to do multipart posts ends up writing their own little library for it, which makes me wonder why this functionality isn't built-in. Maybe it is... Anyway, for your reading pleasure, I'll go ahead and give my solution here. This code is based off of examples I found on a couple of blogs, but I regret that I can't find the links anymore. So I guess I just have to take all the credit for myself...

The module I wrote for this contains one public class, for generating the form data and headers out of a hash of `String` and `File` objects. So for example, if you wanted to post a form with a string parameter named "title" and a file parameter named "document", you would do the following:

```
data, headers = Multipart::Post.prepqre_query("title" => my_string, "document" =>
```

Then you just do a normal `POST` with `Net::HTTP`:

```
http = Net::HTTP.new(upload_uri.host, upload_uri.port)
res = http.start {|con| con.post(upload_uri.path, data, headers) }
```

Or however else you want to do the `POST`. The point is that `Multipart` returns the data and headers that you need to send. And that's it! Simple, right? Here's the code for the Multipart module (you need the `mime-types` gem):

```ruby
# Author:: Cody Brimhall <mailto:cbrimhall@ucdavis.edu>
# Created:: 22 Feb 2008

require 'rubygems'
require 'mime/types'
require 'cgi'


module Multipart
  VERSION = "1.0.0" unless const_defined?(:VERSION)

  # Formats a given hash as a multipart form post
  # If a hash value responds to :string or :read messages, then it is
  # interpreted as a file and processed accordingly; otherwise, it is assumed
  # to be a string
  class Post
    # We have to pretend like we're a web browser...
    USERAGENT = "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/523
    BOUNDARY = "0123456789ABLEWASIEREISAWELBA9876543210" unless const_defined?(:
    CONTENT_TYPE = "multipart/form-data; boundary=#{ BOUNDARY }" unless const_de
    HEADER = { "Content-Type" => CONTENT_TYPE, "User-Agent" => USERAGENT } unles

    def self.prepare_query(params)
      fp = []

      params.each do |k, v|
        # Are we trying to make a file parameter?
        if v.respond_to?(:path) and v.respond_to?(:read) then
          fp.push(FileParam.new(k, v.path, v.read))
        # We must be trying to make a regular parameter
        else
          fp.push(StringParam.new(k, v))
        end
```

edited **Sep 11 '09 at 20:18**

answered **Oct 17 '08 at 18:24**

Cody Brimhall

689 3 13

Hi! What's the license on this code? Also: It might be nice to add the URL for this post in the comments at the top. Thanks! – The Doctor What Sep 23 '10 at 14:34

2 The code in this post is licensed under the WTFPL (sam.zoy.org/wtfpl). Enjoy! – Cody Brimhall Oct 14 '10 at 19:49

you should not pass the filestream into the initialize call of the `FileParam` class. The assignment in the `to_multipart` method copies the file content again, which is unnecessary! Instead pass only the file descriptor and read from it in `to_multipart` – mober Jul 28 at 3:28

feedback

Here is my solution after trying other ones available on this post, I'm using it to upload photo on TwitPic:

```ruby
  def upload(photo)
    `curl -F media=@#{photo.path} -F username=#{@username} -F password=#{@password
  end
```

answered **Dec 26 '08 at 13:09**

Alex

4,504 4 16 24

Despite seeming a bit hackish, this is probably the nicest solution for me so big thanks for this suggestion! – bjeanes Feb 28 '09 at 6:35

Just a note for the unwary, the media=@... is what makes curl thing that ... is a file and not just a string. A bit confusing with ruby syntax, but @#{photo.path} is not the same as #{@photo.path}. This solution is one of the best imho. – Evgeny Feb 22 '10 at 17:48

feedback

Ok, here's a simple example using curb.

```
require 'yaml'
require 'curb'

# prepare post data
post_data = fields_hash.map { |k, v| Curl::PostField.content(k, v.to_s) }
post_data << Curl::PostField.file('file', '/path/to/file'),

# post
c = Curl::Easy.new('http://localhost:3000/foo')
c.multipart_form_post = true
c.http_post(post_data)

# print response
y [c.response_code, c.body_str]
```

edited **Oct 8 '08 at 21:02**

answered **Oct 8 '08 at 18:55**

kch

15.2k 14 62 99

feedback

---

there's also nick sieger's multipart-post to add to the long list of possible solutions.

answered **Apr 23 '09 at 12:26**

Jan Berkel

893 10 8

feedback

---

restclient did not work for me until I overrode create_file_field in RestClient::Payload::Multipart.

It was creating a **'Content-Disposition: multipart/form-data'** in each part where it should be **'Content-Disposition: form-data'**.

http://www.ietf.org/rfc/rfc2388.txt

My fork is here if you need it: git@github.com:kcrawford/rest-client.git

edited **Jan 23 '10 at 5:47**

answered **Jan 14 '10 at 23:05**

user243633

This is fixed in the latest restclient. – user243633 Feb 23 '10 at 18:41

feedback

I had the same problem (need to post to jboss web server). Curb works fine for me, except that it caused ruby to crash (ruby 1.8.7 on ubuntu 8.10) when I use session variables in the code.

I dig into the rest-client docs, could not find indication of multipart support. I tried the rest-client examples above but jboss said the http post is not multipart.

answered **Feb 5 '09 at 19:01**

zd

feedback

Well the solution with NetHttp has a drawback that is when posting big files it loads the whole file into memory first.

After playing a bit with it I came up with the following solution:

```ruby
class Multipart

  def initialize( file_names )
    @file_names = file_names
  end

  def post( to_url )
    boundary = '----RubyMultipartClient' + rand(1000000).to_s + 'ZZZZZ'

    parts = []
    streams = []
    @file_names.each do |param_name, filepath|
      pos = filepath.rindex('/')
      filename = filepath[pos + 1, filepath.length - pos]
      parts << StringPart.new ( "--" + boundary + "\r\n" +
      "Content-Disposition: form-data; name=\"" + param_name.to_s + "\"; filenam
      "Content-Type: video/x-msvideo\r\n\r\n")
      stream = File.open(filepath, "rb")
      streams << stream
      parts << StreamPart.new (stream, File.size(filepath))
    end
    parts << StringPart.new ( "\r\n--" + boundary + "--\r\n" )

    post_stream = MultipartStream.new( parts )

    url = URI.parse( to_url )
    req = Net::HTTP::Post.new(url.path)
    req.content_length = post_stream.size
    req.content_type = 'multipart/form-data; boundary=' + boundary
    req.body_stream = post_stream
    res = Net::HTTP.new(url.host, url.port).start {|http| http.request(req) }

    streams.each do |stream|
```

answered **Dec 25 '08 at 15:46**

Stanislav Vitvitskiy

feedback

Not the answer you're looking for? Browse other questions tagged ruby http post or ask your own question.