What does enctype='multipart/form-data' mean?

Asked 8 years, 10 months ago Active 6 months ago Viewed 925k times

1273

What does `enctype='multipart/form-data'` mean in an HTML form and when should we use it?

html http-headers multipartform-data

edited Feb 2 at 14:21

Mark Amery
74.2k 38 284 334

asked Dec 24 '10 at 12:19

EBAG
7,849 10 49 85

454

w3.org/html/wg/spec/… – JohnOsborne Aug 9 '18 at 10:26

9 Answers

1436

When you make a POST request, 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` (the default)
- `multipart/form-data`
- `text/plain`

Work was being done on adding `application/json`, but that has been abandoned.

(Other encodings are possible with HTTP requests generated using other means than an HTML form submission.)

The specifics of the formats don't matter to most developers. The important points are:

- Never use `text/plain`.

When you are writing client-side code:

- use `multipart/form-data` when your form includes any `<input type="file">` elements
- otherwise you can use `multipart/form-data` or `application/x-www-form-urlencoded` but `application/x-www-form-urlencoded` will be more efficient

When you are writing server-side code:

Most (such as Perl's `CGI->param` or the one exposed by PHP's `$_POST` superglobal) will take care of the differences for you. Don't bother trying to parse the raw input received by the server.

Sometimes you will find a library which can't handle both formats. Node.js's most popular library for handling form data is body-parser which cannot handle multipart requests (but has documentation which recommends some alternatives which can).

If you are writing (or debugging) a library for parsing or generating the raw data, then you need to start worrying about the format. You might also want to know about it for interest's sake.

`application/x-www-form-urlencoded` is more or less the same as a query string on the end of the URL.

`multipart/form-data` is significantly more complicated but it allows entire files to be included in the data. An example of the result can be found in the HTML 4 specification.

`text/plain` is introduced by HTML 5 and is useful only for debugging — from the spec: *They are not reliably interpretable by computer* — and I'd argue that the others combined with tools (like the Net tab in the developer tools of most browsers) are better for that).

edited Apr 12 at 8:04

answered Dec 24 '10 at 12:21

Quentin
692k 82 943 1094

5 @Quentin Excuse me, what will be any probable problem if we use multipart for all forms? with and whit out files. – Webinan Oct 20 '13 at 9:47

9 It doesn't make sense for GET forms, and it makes the file size of requests bigger. – Quentin Oct 20 '13 at 10:46

@Quentin does multipart form data send as a stream by default? – Growler Feb 13 '17 at 23:06

Does the enc in enctype stand for something? – Philip Rego Dec 10 '17 at 21:54

1 "HTML forms provide three methods of **ENC**oding" – Quentin Dec 10 '17 at 22:15

399

> when should we use it

Quentin's answer is right: use `multipart/form-data` if the form contains a file upload, and `application/x-www-form-urlencoded` otherwise, which is the default if you omit `enctype`.

I'm going to:

- add some more HTML5 references
- explain **why** he is right with a form submit example

There are three possibilities for `enctype`:

- `application/x-www-form-urlencoded`
- `multipart/form-data` (spec points to RFC7578)
- `text/plain`. This is "not reliably interpretable by computer", so it should never be used in production, and we will not look further into it.

How to generate the examples

Once you see an example of each method, it becomes obvious how they work, and when you should use each one.

You can produce examples using:

- `nc -l` or an ECHO server: HTTP test server accepting GET/POST requests
- an user agent like a browser or cURL

Save the form to a minimal `.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <title>upload</title>
</head>
<body>
<form action="http://localhost:8000" method="post" enctype="multipart/form-data">
  <p><input type="text" name="text1" value="text default">
  <p><input type="text" name="text2" value="a&#x03C9;b">
  <p><input type="file" name="file1">
  <p><input type="file" name="file2">
  <p><input type="file" name="file3">
  <p><button type="submit">Submit</button>
</form>
</body>
</html>
```

We set the default text value to `a&#x03C9;b`, which means `aωb` because `ω` is `U+03C9`, which are the bytes `61 CF 89 62` in UTF-8.

Create files to upload:

```
echo 'Content of a.txt.' > a.txt

echo '<!DOCTYPE html><title>Content of a.html.</title>' > a.html

# Binary file containing 4 bytes: 'a', 1, 2 and 'b'.
printf 'a\xCF\x89b' > binary
```

Run our little echo server:

```
while true; do printf '' | nc -l 8000 localhost; done
```

Open the HTML on your browser, select the files and click on submit and check the terminal.

multipart/form-data

Firefox sent:

```
POST / HTTP/1.1
[[ Less interesting headers ... ]]
Content-Type: multipart/form-data; boundary=---------------------------
-735323031399963166993862150
Content-Length: 834

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="text1"

text default
-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="text2"

aωb
-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file1"; filename="a.txt"
Content-Type: text/plain

Content of a.txt.

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file2"; filename="a.html"
Content-Type: text/html

<!DOCTYPE html><title>Content of a.html.</title>

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file3"; filename="binary"
Content-Type: application/octet-stream

aωb
-----------------------------735323031399963166993862150--
```

For the binary file and text field, the bytes `61 CF 89 62` (`aωb` in UTF-8) are sent literally. You could verify that with `nc -l localhost 8000 | hd` , which says that the bytes:

```
61 CF 89 62
```

were sent (`61` == 'a' and `62` == 'b').

Therefore it is clear that:

- `Content-Type: multipart/form-data; boundary=---------------------------
-905191404154484336597275426 6` sets the content type to `multipart/form-data` and says that the fields are separated by the given `boundary` string.
- every field gets some sub headers before its data: `Content-Disposition: form-data;` , the field `name` , the `filename` , followed by the data.

  The server reads the data until the next boundary string. The browser must choose a boundary that will not appear in any of the fields, so this is why the boundary may vary between requests.

  Because we have the unique boundary, no encoding of the data is necessary: binary data is

shortest-sequence-that-is-not-a-sub-sequence-of-a-set-of-sequences

- `Content-Type` is automatically determined by the browser.

  How it is determined exactly was asked at: How is mime type of an uploaded file determined by browser?

application/x-www-form-urlencoded

Now change the `enctype` to `application/x-www-form-urlencoded`, reload the browser, and resubmit.

Firefox sent:

```
POST / HTTP/1.1
[[ Less interesting headers ... ]]
Content-Type: application/x-www-form-urlencoded
Content-Length: 51

text1=text+default&text2=a%CF%89b&file1=a.txt&file2=a.html&file3=binary
```

Clearly the file data was not sent, only the basenames. So this cannot be used for files.

As for the text field, we see that usual printable characters like `a` and `b` were sent in one byte, while non-printable ones like `0xCF` and `0x89` took up **3 bytes** each: `%CF%89`!

Comparison

File uploads often contain lots of non-printable characters (e.g. images), while text forms almost never do.

From the examples we have seen that:

- `multipart/form-data`: adds a few bytes of boundary overhead to the message, and must spend some time calculating it, but sends each byte in one byte.
- `application/x-www-form-urlencoded`: has a single byte boundary per field (`&`), but adds a *linear* overhead factor of **3x** for every non-printable character.

Therefore, even if we could send files with `application/x-www-form-urlencoded`, we wouldn't want to, because it is so inefficient.

But for printable characters found in text fields, it does not matter and generates less overhead, so we just use it.

edited Feb 10 at 19:54
VKK
580 3 14

answered Feb 7 '15 at 9:52
Ciro Santilli 新疆改造中心法轮功六四事件
174k 38 665 540

Great post. Question: Why do we have 3 bytes for each non printable character? For eg: for unicode U+03C9, we have %CF%89: this is 2 extra bytes for the two "%". Is my understanding correct? – Khanna111 Aug 6 '15 at 18:42

3 @Khanna111 `%CF` is 3 bytes long: `%`, `C` and `F` :-) Story of making it human readable. –

5 On OS X, `nc` won't accept both the `-l` and the `-p` arguments simultaneously. But this works for me: `while true; do printf '' | nc -l 8000; done` . – PhilipS Apr 25 '17 at 2:03

3 A small but important point that isn't mentioned is that the boundary specified in the `Content-Type` has two hyphens (`--`) less, i.e. when actually using the boundary in the message body, you must prefix it with `--` . Also, the last boundary must be suffixed with `--` , but that is easy enough to notice. See stackoverflow.com/questions/3508252/… – Bernard Jul 3 '17 at 12:34

85

`enctype='multipart/form-data` is an encoding type that allows files to be sent through a *POST*. Quite simply, without this encoding the files cannot be sent through *POST*.

If you want to allow a user to upload a file via a form, you must use this *enctype*.

edited Dec 12 '12 at 11:30

Habeeb Perwad
3,518 12 66 113

answered Dec 24 '10 at 12:49

Matt Asbury
4,987 18 28

So.. if the file is not a binary file then can we work without this ? – Yugal Jindle Aug 27 '13 at 0:01

From what I understand, you can use `multipart/form-data` for sending non-binary files but it is inefficient. I believe using `application/x-www-form-urlencoded` is the correct way to send non-binary data but someone with more experience with non-binary files may need to correct me. – Matt Asbury Aug 27 '13 at 9:36

9 The main advantage to using `multipart/form-data` for sending a file is that it will work automatically in both frontend and backend. You don't have to do any special handling. All files are binary even if they should only contain text. `application/x-www-form-urlencoded` is the standard way to POST a form without attached files. `multipart/form-data` is the standard way to POST a form with attached file(s). (There are also numerous other encodings, such as `application/json` and `application/json-patch+json` , which are common for communication between server and client.) – Daniel Luna Sep 19 '13 at 17:34

5 Its worth pointing out you can base64 encode your image and send it as plain string data . – Arcabard Jul 14 '14 at 22:46

2 Further to @Prospero's comment above: you can absolutely send files via POST without using `multipart/form-data` . What you *can't* do is do that using an ordinary HTML form submission, without JavaScript. Setting a form to use `multipart/form-data` is the only mechanism that *HTML* provides to let you POST files without using JavaScript. I feel like this isn't clear enough in the answer, and that a naive reader might think that the inability to send files without `multipart/form-data` is a limitation of *HTTP*; that's not the case. – Mark Amery Feb 2 at 14:29

73

When submitting a form, you tell your browser to send, via the HTTP protocol, a message on the network, properly enveloped in a TCP/IP protocol message structure. An HTML page has a way to send data to the server: by using `<form>` s.

When a form is submitted, an HTTP Request if created and sent to the server, the message will contain the field names in the form and the values filled in by the user. This transmission can happen with `POST` or `GET` *HTTP methods*.

- `POST` tells your browser to build an HTTP message and put all content in the body of the message (a very useful way of doing things, more safe and also flexible).
- `GET` will submit the form data in the *querystring*. It has some constraints about data

Attribute `enctype` has sense only when using `POST` method. When specified, it instructs the browser to send the form by encoding its content in a specific way. From MDN - Form enctype:

> When the value of the method attribute is post, enctype is the MIME type of content that is used to submit the form to the server.

- `application/x-www-form-urlencoded` : This is the default. When the form is sent, all names and values are collected and URL Encoding is performed on the final string.
- `multipart/form-data` : Characters are NOT encoded. This is important when the form has a file upload control. You want to send the file binary and this ensures that bitstream is not altered.
- `text/plain` : Spaces get converted, but no more encoding is performed.

Security

When submitting forms, some security concerns can arise as stated in RFC 7578 Section 7: Multipart form data - Security considerations:

> All form-processing software should treat user supplied form-data with sensitivity, as it often contains confidential or personally identifying information. There is widespread use of form "auto-fill" features in web browsers; these might be used to trick users to unknowingly send confidential information when completing otherwise innocuous tasks. multipart/form-data does not supply any features for checking integrity, ensuring confidentiality, avoiding user confusion, or other security features; those concerns must be addressed by the form-filling and form-data-interpreting applications.
>
> Applications that receive forms and process them must be careful not to supply data back to the requesting form-processing site that was not intended to be sent.
>
> It is important when interpreting the filename of the Content-Disposition header field to not inadvertently overwrite files in the recipient's file space.

This concerns you if you are a developer and your server will process forms submitted by users which might end up containing sensitive information.

edited Feb 3 at 9:25

answered Dec 24 '10 at 17:50

Andry
7,292 21 99 191

1 The stuff about security after the most recent edit is all irrelevant to the question of what `enctype` s do. I know it's literally from the `multipart/form-data` RFC, but nonetheless it's an arbitrary dump of security considerations about submitting forms that are entirely orthogonal to whether data is sent as `application/x-www-form-urlencoded` or `multipart/form-data` . – Mark Amery Feb 3 at 9:51

`enctype='multipart/form-data'` means that no characters will be encoded. that is why this type is used while uploading files to server

edited Oct 13 '14 at 21:09

Volker E.
4,590 10 39 62

answered Jul 4 '13 at 9:13

GP Singh
477 4 11

8

Set the method attribute to POST because file content can't be put inside a URL parameter using a form.

Set the value of enctype to multipart/form-data because the data will be split into multiple parts, one for each file plus one for the text of the form body that may be sent with them.

answered Sep 25 '13 at 11:53

sandy
358 3 10

This implies that `POST` is likely to be sufficient for submitting a file via a form and that adding `multipart/form-data` is just a bonus in some vague way. That's not the case. Most files will absolutely require using `multipart/form-data` . – underscore_d Jun 11 '17 at 12:33

0

Usually this is when you have a POST form which needs to take a file upload as data... this will tell the server how it will encode the data transferred, in such case it won't get encoded because it will just transfer and upload the files to the server, Like for example when uploading an image or a pdf

answered Mar 10 '16 at 9:29

Eric
130 1 15

0

- enctype(**ENC**ode **TYPE**) attribute specifies how the form-data should be encoded when submitting it to the server.
- **multipart/form-data** is one of the value of enctype attribute, which is used in form element that have a file upload. **multi-part** means form data divides into **multiple parts** and send to server.

edited Feb 3 at 4:00

answered Dec 27 '15 at 1:29

Premraj
39k 18 180 131

4 I believe **enctype** does not stand for encryption type. There is no encryption involved at this level. My guess is either encoding type or enclosed type. But surely it is not encryption type. – Yeo Mar 3 '16 at 11:00

Your final bullet point here about `<head>` and `<body>` is irrelevant and confusing. – Mark Amery Feb 2 at 14:55

> No characters are encoded. This value is required when you are using forms that have a file upload control

From W3Schools

edited Nov 14 '18 at 2:25

Pang
7,274 16 68 108

answered Nov 11 '18 at 22:52

Rishad
173 1 10

This quote doesn't even mention `multipart/form-data` . It's also pretty unclear; what does the sentence "No characters are encoded" even mean? -1. – Mark Amery Feb 2 at 14:22

protected by NINCOMPOOP Nov 4 '13 at 7:17

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?