

# Summary of the accept-charset attribute

Have you ever noticed some characters on a site not displaying correctly? Maybe the quotation marks look like little white boxes, or the long dashes have been replaced with question marks ...

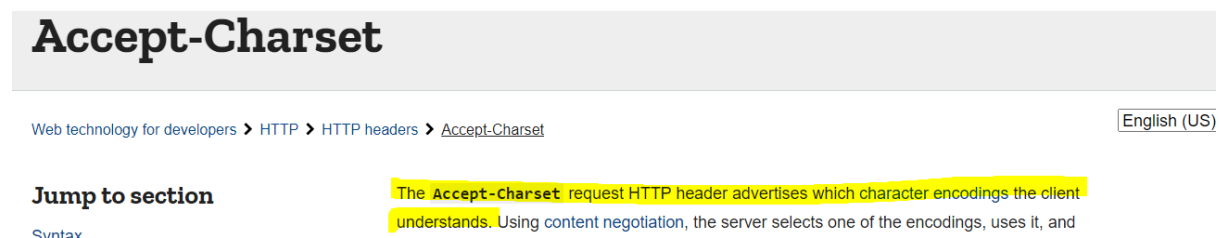
Well, problems like these usually arise from little understanding of character encodings on the part of the developer responsible for the site.

Remember, by ‘encoding’ I mean the process of transforming characters from one format into another. It’s not magic. It’s not difficult and having a good understanding of the topic will help you *a lot* in your programming career.

## What does encoding have to do with FORMS?

When a form is submitted to a server, the browser uses the character sets to convert the text content of the form to a byte stream that the server can accept. You can also think of the `accept-charset` as the browsers way of telling the server what character encodings the browser understands.

Indeed, this is exactly what MDN says, as you can see from the screenshot below:



As we have seen, if the encoding type you specify does not include the character the user has typed into the form, the browser will attempt to covert this number into its own character set format.

This format may or may not be understood by the server, which is why it poses a problem.

As mentioned in the previous lecture, always use UTF-8 unless you have a very good reason not to.


## How does the browser use the `accept-charset` attribute?

What a lot of people don't know is that the browser USED to send `accept-charset` parameters in HTTP request headers according to their own principles and settings (regardless of your accept-content type).

But today, this header is often ignored by server-side code. And what's even more, browsers don't even send it anymore.

Again, MDN confirms this:

## Accept-Charset

 **Warning:** Do not use this header. Browsers omit this header and servers should ignore it.

The `Accept-Charset` request HTTP header was a header that advertised a client's supported [character encodings](#). It is no longer widely used.

UTF-8 is well-supported and the overwhelmingly preferred choice for character encoding. To [guarantee better privacy through less configuration-based entropy](#), all browsers omit the `Accept-Charset` header. Chrome, Firefox, Internet Explorer, Opera, and Safari abandoned this header.

Today, `Accept-Charset` is most notable for being one of several [forbidden header names](#).

accept-charset header

Remember, the `accept-charset` attribute was designed for 2 main reasons:

1. to make the encoding of the FORM data submission different from the encoding of the page. Suppose your page is ASCII encoded and someone type's Hebrew or Greek into your form. Browsers will have to figure out what this means, since those Hebrew or Greek characters can't be presented in ASCII format; and
2. so that the server can determine which encoding is to be used in its response. For example, server-side software (a form handler, in this case) may be capable of using different encodings in its response.

So then, you may be wondering, do I need to explicitly use the `accept-charset` attribute?

**Short answer: no.**

Why?

Firstly, the default character set of your entire form is the same as the character set of the entire document. So, the only time you would need to define the `accept-charset` attribute is if any element of the form contains characters that can't be represented by the character set of the document.

Secondly, for HTML5, the default character encoding is now UTF-8. This was not always the case. The default character encoding for the early web was ASCII. But with the introduction of XML and HTML5, UTF-8 became the default and has solved a lot of problems.

And thirdly, servers typically ignore it, and because of these browsers no longer said this header information along with a request.

## Conclusion

I know I spent some time on this, but it's so important for you to grasp these concepts early on in your career. It will only help you become a better programmer. We're also going to talk about the Content-Type (or more specifically, the `enctype` attribute) later on, and having this background information will help you.

By the way, here is a cool [link](#) that lets you play around with bytes and encoding types.

See you in the next lecture :)