

## Handling Character Encodings

Anyone who's ever written a form for user input and actually cares about ensuring the correct character encoding is submitted has had trouble with users submitting Windows-1252, where ISO-8859-1 was expected. Even if you were intelligent and were using a Unicode encoding like UTF-8 and accepting such input from your forms, there's still a problem with Trackbacks, since you can't have no control over what encoding they're sent in.

This is commonly ignored by implementations and results in invalid characters used within HTML and you end up a few question marks (commonly shown as a U+FFFD Replacement Character by browsers) scattered around the text.

Now there is a solution. I've written some PHP to first detect the most likely encoding as either being UTF-8, ISO-8859-1 or Windows-1252. If it is UTF-8, nothing needs to be done with it. If it's ISO-8859-1 or Windows-1252, we need to convert it to UTF-8.

### Determining the Encoding

The first 3 functions I've written will allow you to determine what character encoding is used. These are `isUTF8()`, `isISO88591()` and `isCP1252()` and return true if the string validates as the respective encoding. These work by using regular expression that matches valid octet sequences for the encoding. The regular expression for UTF-8 was adapted from the Perl code provided by the W3C in an article about multilingual forms.

My version is a little more restrictive than that, in that it will reject any character with a code point from 128 to 159. Although these code points are valid in XML and can be validly encoded in UTF-8, they are Unicode control characters and they are invalid within HTML 4. Additionally, the chances of a user legitimately submitting those characters are slim to nil, so it's better to reject them than try to convert them to something else.

The ISO-8859-1 function works in the same way. It too rejects characters with those code points, as it is far more likely that the user has submitted Windows-1252 than the control characters.

### Converting to UTF-8

In PHP, the `utf8_encode()` function can be used to convert from ISO-8859-1 to UTF-8. However, the real world forces us to handle ISO-8859-1 as Windows-1252, yet the `utf8_encode()` function will not handle that as well as we would like.

Since Windows-1252 is a superset of ISO-8859-1, these can both be handled by the same function: `utf8FromCP1252()`. Internally, this makes use of the pre-existing `utf8_encode()` function. Afterwards, it searches the newly encoded UTF-8 string for characters in the offending code points and remaps them to their correct Unicode code points and encodes them.

To do this a second function is used which accepts the Windows-1252 encoded character, determines the code point, uses a look up table in an array to find the Unicode code point and then calls a third function to generate the UTF-8 encoded character from that code point.

The third function has been adapted from Anne Van Kesteren's Character references to UTF-8 converter, who originally adapted it from Henri Sivonen's UTF-8 to Code Point Array Converter. The main difference with my version is that I renamed it and changed the variable names used to something a little more sensible.

### Code and Demo

You can see it all in action on the demonstration page. Enter some characters in the UTF-8 for and the ISO-8859-1 forms and see how it flawlessly handles the detection and conversion of your input into valid UTF-8 output. The source code is available also.

Filed under [Server-Side](#), [Characters](#).