# [The entity “nbsp” was referenced, but not declared](http://stackoverflow.com/questions/13012327/error-parsing-page-xhtml-error-tracedline-42-the-entity-nbsp-was-referenc)

Facelets is a XML based view technology. XML has only [five predefined entities](http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references#Predefined_entities_in_XML). The &nbsp; is not among them. It works only when used in plain HTML or in legacy JSP (note: it doesn't work in JSPX as that's also XML based!).

To fix this, you either need to declare the entity yourself in the doctype,

<!DOCTYPE html [

<!ENTITY nbsp "&#160;">

]>

or to use the (hexa)decimal notation instead: &#xA0; or &#160;

# Predefined entities in XML[[edit](http://en.wikipedia.org/w/index.php?title=List_of_XML_and_HTML_character_entity_references&action=edit&section=3" \o "Edit section: Predefined entities in XML)]

The XML specification does not use the term "character entity" or "character entity reference". The XML specification defines five "predefined entities" representing special characters, and requires that all XML processors honor them. The entities can be explicitly declared in a DTD, as well, but if this is done, the replacement text must be the same as the built-in definitions. XML also allows other named entities of any size to be defined on a per-document basis.

The table below lists the five XML predefined entities. The "Name" column mentions the entity's name. The "Character" column shows the character. To render the character, the format &name; is used; for example,&amp; renders as &. The "Unicode code point" column cites the character via standard UCS/Unicode "U+" notation, which shows the character's code point in hexadecimal. The decimal equivalent of the code point is then shown in parentheses. The "Standard" column indicates the first version of XML that includes the entity. The "Description" column cites the character via its canonical UCS/Unicode name, in English.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Character** | **Unicode code point (decimal)** | **Standard** | **Description** |
| quot | " | U+0022 (34) | XML 1.0 | double [quotation mark](http://en.wikipedia.org/wiki/Quotation_mark) |
| amp | & | U+0026 (38) | XML 1.0 | [ampersand](http://en.wikipedia.org/wiki/Ampersand) |
| apos | ' | U+0027 (39) | XML 1.0 | [apostrophe](http://en.wikipedia.org/wiki/Apostrophe) *(apostrophe-quote)* |
| lt | < | U+003C (60) | XML 1.0 | [less-than sign](http://en.wikipedia.org/wiki/Less-than_sign) |
| gt | > | U+003E (62) | XML 1.0 | [greater-than sign](http://en.wikipedia.org/wiki/Greater-than_sign) |

All answers posted so far are giving the right solutions, however no one answer was able to properly explain the underlying cause of the concrete problem.

Facelets is a XML based view technology which uses XHTML+XML to generate HTML output. XML has five special characters which has special treatment by the XML parser:

* < the start of a tag.
* > the end of a tag.
* " the start and end of an attribute value.
* ' the alternative start and end of an attribute value.
* & the start of an entity (which ends with ;).

In case of & which is not followed by # (e.g. &#160; or &#xA0;), the XML parser is implicitly looking for one of the [predefined entity names](http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references#Predefined_entities_in_XML) such as lt, gt, amp, etc. However, in your particular case, you was using & as a JavaScript operator, not as an XML entity. This totally explains the XML parsing error you got:

The entity name must immediately follow the '&' in the entity reference

In essence, you're writing JavaScript code in the wrong place, a XML document instead of a JS file, so you should be escaping all XML special characters accordingly. The & must be escaped as &amp;.

So, in your particular case, the

if (Modernizr.canvas && Modernizr.localstorage &&

must become

if (Modernizr.canvas &amp;&amp; Modernizr.localstorage &amp;&amp;

to make it XML-valid.

However, this makes the JavaScript code harder to read and maintain. As stated in Mozilla Developer Network's excellent document [Writing JavaScript for XHTML](https://developer.mozilla.org/en/docs/Writing_JavaScript_for_XHTML), you should be placing the JavaScript code in a character data (CDATA) block. Thus, in JSF terms, that would be:

<h:outputScript>

<![CDATA[

// ...

]]>

</h:outputScript>

The XML parser will interpret the block's contents as "plain vanilla" character data and not as XML and hence interpret the XML special characters "as-is".

But, much better is to just put the JS code in its own JS file which you include by <script src>, or in JSF terms, the <h:outputScript>.

<h:outputScript name="onload.js" target="body" />

(note the*target="body"*; this way JSF will automatically render the*<script>*at the very end of*<body>*, regardless of where*<h:outputScript>*itself is located, hereby achieving the same effect as with*window.onload*and*$(document).ready()*; so you don't need to use those anymore in that script)

This way you don't need to worry about XML-special characters in your JS code. As an additional bonus, this gives you the opportunity to let the browser cache the JS file so that total response size is smaller.

### See also:

* [the entity nbsp was referenced but not declared](http://stackoverflow.com/questions/13012327/the-entity-nbsp-was-referenced-but-not-declared/)
* [Is it possible to use JSF+Facelets with HTML 4/5?](http://stackoverflow.com/questions/2935759/is-it-possible-to-use-jsffacelets-with-html-4-5)
* [How to reference resource in Facelets template?](http://stackoverflow.com/questions/8367421/how-to-reference-resource-in-facelets-template)
* [Writing JavaScript for XHTML](https://developer.mozilla.org/en/docs/Writing_JavaScript_for_XHTML)

Change to upper case letter when user jump to another component. There are indeed 2 ways to salvage this.

1. Using JavaScript.

<h:inputText ... onblur="value=value.toUpperCase()" />

1. Using JSF.
2. <h:inputText ... converter="toUpperCaseConverter">
3. <f:ajax event="blur" render="@this" />

</h:inputText>

with something like

@FacesConverter("toUpperCaseConverter")

public class ToUpperCaseConverter implements Converter {

@Override

public String getAsString(FacesContext context, UIComponent component, Object value) {

return (String) value; // Or (value != null) ? value.toString().toUpperCase() : null;

}

@Override

public Object getAsObject(FacesContext context, UIComponent component, String value) {

return (value != null) ? value.toUpperCase() : null;

}

}

The JS approach is extremely simple. However, this is tamperable by the enduser as it's performed fully at the client side, under full control of the enduser. The enduser can disable/skip that JS code and/or edit the request parameter before it's actually being sent to the server side. The JSF approach isn't tamperable as this is performed fully at the server side, so this results in a more solid and reliable result.

You have to decide based on those facts which one fits the business requirements the best.