

Escaping

How to properly escape html/js code in Smarty templates and PHP files

- [Introduction](#)
- [Escaping rules](#)
- [Cheat sheet](#)

Introduction

Starting with vtenext version 26.01, all variables outputted by Smarty templates (for example `{ $VARIABLE }`) by default have all applicable characters are converted to html `&...;` notation. For instance, the string `"nice & 'smooth' à > è"` is converted to `"nice & &039;smooth&039; à > è"`

The conversion is done with the `htmlentities` function, without double encoding existing entities.

The reason of this change is to reduce as much as possible exposure to XSS attacks, since it was extremely difficult to track every possible variable in templates and ensure it was properly escaped.

There are of course some exceptions on the escaping and some edge cases that should be understood. The next chapter presents the rules to follow to write simple and secure code.

Escaping rules

1. Avoid html code in php

Avoid generating html code directly in php, or outputting it directly. If possible use Smarty templates. Exceptions can be made for very short snippets, see paragraph 3 for how to handle them.

Examples

Good

```
$name = "John & friends";  
$smarty->assign("NAME", $name);
```

And in smarty:

```
{* & will be converted to &amp; *}  
<div>{$NAME}</div>
```

Will output:

```
<div>John &amp; friends</div>
```

Bad: html generated in php

```
$name = "John & friends";  
// the "div" will be escaped  
$string = "<div>{$name}</div>";  
$smarty->assign("TEXT", $string);
```

And in smarty:

```
{ $TEXT }
```

Will output:

```
&lt;div&gt;John &amp; friends&lt;/div&gt;
```

Bad: output via echo

```
$name = "John & friends";  
$string = "<div>{$name}</div>";  
  
// by using echo on the raw string,  
// we introduce a XSS if $name comes from  
user input  
  
echo $string;
```

2. Reading from database

Avoid using `query_result` and `fetchByAssoc` without the `-1, false` 2nd and 3rd parameters.

This is because the those functions already do html conversion, which is the wrong place to do it, since the result may have to be processed more before being displayed by the browser (or not displayed at all in a browser if the result is for a REST API).

Examples

Good <pre data-bbox="140 286 734 515"> \$res = \$adb->query("SELECT helpinfo FROM vte_field"); \$first = \$adb->query_result_no_html(\$res, 0, 'helpinfo');</pre>	Bad: helpinfo is converted! <pre data-bbox="842 286 1436 515"> \$res = \$adb->query("SELECT helpinfo FROM vte_field"); \$first = \$adb->query_result(\$res, 0, 'helpinfo');</pre>
Good <pre data-bbox="140 772 734 1093"> \$res = \$adb->query("SELECT helpinfo FROM vte_field"); while (\$row = \$adb->fetchByAssoc(\$res, -1, false)) { // ... }</pre>	Bad: helpinfo is converted! <pre data-bbox="842 772 1436 996"> \$res = \$adb->query("SELECT helpinfo FROM vte_field"); while (\$row = \$adb->fetchByAssoc(\$res)) { // ... }</pre>

3. Generating html in php

Sometimes it's really inevitable to generate html code in php (legacy code, or existing developments...), so in this case, to avoid a double conversion when Smarty processes the template, we should use the `\Vtenext\Types\HtmlString` class to wrap the html string. This class is a simple wrapper around a string, that doesn't get automatically converted by Smarty. Remember, that in this case, all the html conversion **must be done** in php using the `VStr::toHtml` or `VStr::toHtmlAttr` methods.

Example:

```

// for brevity:
use \Vtenext\Types\HtmlString;

$name = "John Connor";
$url = "index.php?module=aaa&action=".urlencode($_REQUEST['param']); // danger here: user input!
```

```
// building the link with manual conversion
$link = "<a href='".VStr::toHtmlAttr($url)."'>".VStr::toHtml($name)."</a>";

// wrapping it in HtmlString
$link = new HtmlString($link);

$smarty->assign("LINK", $link); // this will NOT be escaped!! Be sure all parameters are
properly escaped!!
```

There are also 2 utility methods to build html string safely, `HtmlString::build` and `HtmlString::buildSmarty`. Let's see some examples:

```
// with build(), every instance of ### is replaced with one of the parameters,
// and escaped for html. The resulting string is safe for raw inclusion in templates
$icon = HtmlString::build('<i class="vteicon md-sm" title="###" style="color:
white;">settings</i>', [trans('LBL_AREAS_SETTINGS')]);

# even when reading from $_REQUEST, we don't have XSS here
$errorStr = HtmlString::build("<font class='error'>Error: ###</font>", RH::r('error_string'));

// buildSmarty takes a smarty template in string form and does the standard conversion:
$tpl = '
    <input class="crmbutton" onclick="myFunction(\'{$module}\', {$recordid}, \'{$entityName}\')"
type="button"
    value="{ "LBL_LINK_ACTION" |trans}">';

$params = [
    'module' => $currentModule,
    'recordid' => intval($recordid),
    'entityName' => VStr::toJs($entityName, ''), // js stuff must be explicitly escaped
];
$html = HtmlString::buildSmarty($tpl, $params);
```

4. Assigning to Smarty

Assign the variables normally, without using `decode_html`, `htmlspecialchars`, `htmlentities` or other conversion functions.

Examples

Good

```
$string = "my nice text with < and >";  
$smarty->assign("VAR", $string);
```

And in smarty:

```
{* < and > will be converted *}  
<div>{$VAR}</div>
```

Bad: no need to convert 2 times

```
$string = "my nice text with < and >";  
$smarty->assign("VAR",  
    htmlentities($string));
```

And in smarty:

```
{* still work the same,  
since double_encoding is false  
but not a good idea anyway *}  
<div>{$VAR}</div>
```

5. Inside Smarty templates (html)

Output variables as they are, no additional escaping needed in html. For variables inside javascript code (ex: in onclick attributes) see the next paragraph.

6. Inside Smarty templates (javascript)

Javascript code, inside Smarty templates, delimited by `<script>` tags, has a special handling. Inside these blocks, the default html conversion is not done, instead, the following happens:

1. If the variable to replace begins with `[` or `{`, no escaping is done
2. Otherwise, the variable is escaped with `vStr::toJs` (which does a addslashes)

For examples, if in PHP we have:

```
$arr = [1,2,3];  
$obj = ['name' => 'test'];  
  
// no need of js escaping here  
$smarty->assign("STR", "test <tag> 'quote' end");
```

```
// this is the preferred way to pass complex types, as they are!
$smarty->assign("ARR", $arr);
$smarty->assign("OBJ", $obj);

// for legacy code it's acceptable to have this
$smarty->assign("JSARR", json_encode($arr));
$smarty->assign("JSOBJ", json_encode($obj));

// AVOID to build js structures with string manipulation in PHP!!
```

then in the template we can use:

```
<script>
    // both double quotes and single quotes can be used, they are both escaped!
    var str1 = '{$STR}'; // will become test <tag> \'quote\' end
    var str2 = "{$STR}";

    // the preferred way is to use json_encode for complex types
    var arr = {$ARR|json_encode};
    var obj = {$OBJ|json_encode};

    // but in case something is already in json form:
    var arr = {$JSARR}; // this WILL NOT be escaped!
    var obj = {$JSOBJ}; // also this one

</script>
```

Outside of `<script>` tags, the standard html conversion is done, so in case of javascript code in attributes (ex: onclick handlers), manual escaping is still necessary, for example:

```
{* using escape in "javascript" mode *}
<span onclick="myFunction('{$PARAM|escape:"javascript"}')">Link</span>

{* using out VStr::toJs method *}
<span onclick="myFunction2('{VStr::toJs($PARAM)}')">Link2 </span>

{* for urls inside js, use "url" mode before the js encoding *}
<span
onclick="location.href='index.php?mode={$MODPARAM|escape:"url"|escape:"javascript"}';">Link
```

7. Skipping the conversion

Sometimes it's necessary to override the default conversion and output the raw variable as is. This can be achieved in two ways:

1. By using `nofilter`:

```
<span>{$RAW_VAR nofilter}</span>
```

2. By using the modifier `rawhtml` (will convert the string to `HtmlString`, thus avoiding the conversion)

```
<span>{$RAW_VAR|rawhtml}</span>
```

8. Special cases: labels

When using labels in templates (ex: `"LBL_SOMETHING"|trans` or `$APP.LBL_SOMETHING`), html chars are converted. Some labels, though, contains html code, and in this case they should be used with the `rawhtml` or `nofilter` modifiers.

9. Special cases: {capture}

When using `{capture}` in templates, the content is saved in a variable. If that variable is then outputted, the content is escaped, which is usually unwanted since the content is html. In this case, the `rawhtml` or `nofilter` modifier must be used.

For example, if the capture block is:

```
{capture assign="content"}  
<div> Hello {$FRIEND}</div> {* $FRIEND will be converted to html entities *}  
{/capture}
```

It should be used with:

```
<h2>{$content|rawhtml}</h2>
```

or

```
<h2>{$content nofilter}</h2>
```

or

```
{include file="NoLoginMsg.tpl" BODY=$content|rawhtml}
```

Cheat sheet

Quick reference if you just need to know how to escape stuff!

Do's and Don'ts

- Use only `query_result_no_html` and `fetchByAssoc(..., -1, false)` or `fetchByAssocNoHtml`, we don't want that pesky `to_html` function to be called
- Do not generate html strings in php, use templates or `HtmlString::build`
- Do not echo html code in PHP, use Smarty
- Do not use `VStr::toJsAttr` method
- Do not try to build js code from PHP, use .js files or `<script>` tags in templates
- Do not use `html_entity_decode`, `htmlentities`, `htmlspecialchars`, `addslashes`, it's probably not needed (unless you are working on legacy code)
- Do not use `to_html`, `from_html`, `decode_html`, these were always a bad idea

How to's:

In	How do I handle... ?	... like this:
Smarty, html code	standard variable	<code>{ \$VARIABLE }</code>
	variable, but it's a <code>HtmlString</code>	<code>{ \$VARIABLE }</code>
	variable, but it's a string and already html	<code>{ \$VARIABLE nofilter }</code> or <code>{ \$VARIABLE rawhtml }</code>

In	How do I handle... ?	... like this:
<code>{capture}</code> blocks	<pre> {capture assign="capname"} <div>.... html code {\$VARIABLE} </div> {capture} {\$capname nofilter} </pre>	
Smarty, inside <code><script></code>	string variable object or array variable string inside url	<pre>var myvar = '{\$VARIABLE}';</pre> <pre>var mylist = {\$VARIABLE json_encode};</pre> <pre>var url = "index.php?module={\$VARIABLE escape:"u rl"}";</pre>
Smarty, js in attributes	string variable	<pre> {* using escape in "javascript" mode *} L ink {* using out VStr::toJs method *} Link2 </pre>

In	How do I handle... ?	... like this:
string in url	<pre>Link 3</pre>	