



## Moodle Enhancement Proposal - Enhancement of javascript / css linking for modules / blocks and filters

### **Background**

Many 3<sup>rd</sup> party modules, blocks and filters which make use of javascript or css have trouble inserting the links into the head of moodle's html document (between <head></head>).

This has lead to a number of 3<sup>rd</sup> party enhancements which break XHTML standards compliance and best practices, e.g. files linked inside the body of the html document, or even worse before the <html> tag! With javascript becoming more heavily used with web 2.0 style development (e.g. AJAX), this problem really needs to be addressed.

Currently, it is possible to include javascript (but not css) by adding a file named 'javascript.php' to the module, block or filter.

What does this do and why does it not work? Adding a javascript.php file to your block returns javascript that is then inserted into the head of the html doc generated by Moodle. The file extension of javascript.php implies that you can dynamically generate javascript according to various variables (e.g. user capabilities, etc). However, the javascript.php file is simply treated as a text file and everything (including your PHP code!) is inserted into the head of your html document!

This problem is caused by javascript-mod.php which simply recovers the unparsed contents of the 'javascript.php' file from each module, block or filter- e.g.

```
if ($mods = get_list_of_plugins('mod')) {
    foreach ($mods as $mod) {
        if (is_readable($CFG->dirroot.'/mod/'.$mod.'/javascript.php')) {
            $output .= file_get_contents($CFG->dirroot.'/mod/'.$
$mod.'/javascript.php');
        }
    }
}
```



Also, another problem arises from the fact that javascript-mod.php recovers script from EVERY filter, module and block regardless of whether its hidden or if an instance of it currently exists on the page! This is a very band-width wasteful way to provide script inserts into the head of the html document. For example, a block with code specifically for teachers would also be loaded for students. You could also argue that this is a mild security risk because the students should not be aware of js code they do not need to see (even if the server php code ensures security).

### ***Proposed Solution***

To avoid breaking compatibility with old templates and 3<sup>rd</sup> party modules, I propose that the moodle/lib/javascript.php file is modified to include the rendered output of 2 new optional module files. These 2 new optional files could be added to either a module, block or filter directory:

moodle/blocks/css\_js\_example/header\_ **js.php**

moodle/blocks/css\_js\_example/header\_ **css.php**

Both header\_js.php and header\_css.php will provide inline and linked files.

### **Example of header\_js.php placed within a block:**

```
<?php
global $CFG, $COURSE, $HEADINLINE, $HEADLINKS; // note, both $HEADINLINE and
$HEADLINKS are wiped each time before htmlheadlib.php processes a module's
inserts. This means, hackers cannot use these global variables to inject code
into the html document header.

// Do not include script tags (they will be added automatically for you)
$HEADINLINE=""
alert ('test');
";

// Array of linked javascript files (urls)
$HEADLINKS=array();

// Capabilities not set - everyone gets this javascript
```



```

$HEADLINKS[]=array(
    'src'=>$CFG->wwwroot.'/blocks/head_ins_example/js/main.js'
);

// Capabilities set - only people with access to update course get this javascript
$HEADLINKS[]=array(
    'src'=>$CFG->wwwroot.'/blocks/head_ins_example/js/teacher.js',
    'caps'=>array(
        array(
            'instanceid'=>$COURSE->id,
            'capability'=>'moodle/course:update',
            'capcontext'=>CONTEXT_COURSE
        )
    )
);

// If you want to include multiple sources against 1 set of conditions (e.g.
// capabilities) then simply make the src property an array
$HEADLINKS[]=array(
    'src'=>array(
        $CFG->wwwroot.'/blocks/head_ins_example/js/script1.js',
        $CFG->wwwroot.'/blocks/head_ins_example/js/script2.js',
        $CFG->wwwroot.'/blocks/head_ins_example/js/script3.js'
    ),
    'caps'=>array(
        array(
            'instanceid'=>$COURSE->id,
            'capability'=>'moodle/course:update',
            'capcontext'=>CONTEXT_COURSE
        )
    ),
    'mode'=>'edit'
);

?>

```

## ***Creating your own header\_js / header\_css file***

Your header\_js or header\_css file must be located in the root of your block, filter or module folder.

Your file must begin with the following code:

```

<?php
global $CFG, $COURSE, $HEADINLINE, $HEADLINKS; // note, both $HEADINLINE and
$HEADLINKS are wiped each time before htmlheadlib.php processes a module's
inserts. This means, hackers cannot use these global variables to inject code
into the html document header.

```



To add inline javascript/css to the head, simply set the \$HEADINLINE variable to some script/css. NOTE: You do not have to add script or style tags as they are automatically added for you.

```
$HEADINLINE=""  
alert ('test');  
";
```

Note: The \$HEADINLINE variable does not offer any capability, mode or browser checking.

To add linked javascript/css to the head you will need to add items to the \$HEADLINKS.

Each array item within \$HEADLINKS should also be an array comprising at least the 'src' key.

E.G.

```
$HEADLINKS[]=array(  
    'src'=>'http://www.arandomsite.com/js/test.js'  
);
```

Note, you can also make src an array to include a group of scripts / css

E.G.

```
$HEADLINKS[]=array(  
    'src'=>array(  
        'http://www.arandomsite.com/js/test.js',  
        'http://www.arandomsite.com/js/test2.js'  
    )  
);
```



The optional keys within a HEADLINK item are as follows:

**caps** – an array of capabilities that permit the script / css to load

```
'caps'=>array(
  array(
    'instanceid'=>${COURSE->id},
    'capability'=>'moodle/course:update',
    'capcontext'=>CONTEXT_COURSE
  ),
  array(
    'capability'=>'block/head_ins_example:testblockcap',
    'capcontext'=>CONTEXT_BLOCK
  )
)
```

**note:**

Each caps item is an array which must contain the 'capability' key string. The capability key should contain a capability in the standard moodle format – e.g. 'moodle/course:update'

The 'instanceid' key is optional. If you set 'capcontext' to CONTEXT\_COURSE (or omit the 'capcontext' key) it will use the current courses instanceid (or site instanceid if you are on the front page).

If you set 'capcontext' to CONTEXT\_BLOCK, the instanceid will be retrieved from the block instance

The 'capcontext' key is optional – it will default to CONTEXT\_COURSE

**mode** – a string indicating the mode required for the script / css to load – 'edit', 'read' or 'both'. Note: Omitting this key will load the script for both modes

```
'mode'=>'edit'
```

**browsers** – an array of browsers that require the script / css. Note that each browser item is an array containing a 'brand' key and a 'version' key:

```
'browsers'=>array(array('brand'=>'Firefox', 'version'=>0))
```



## Testing

Applying the core hack:

Copy the contents of the 'moodle' folder into your test moodle installations directory. This will also copy the example block 'head\_ins\_example' (short for head insert example).

For a crude but workable example of how dynamic css/js can be added via a block, view the javascript and css in your html head – then add the 'head\_ins\_example' block to your front page and view the js and css in your html head. You should notice that the following javascript and css have been dynamically added to the html head by the block:

```
<!--Start of script generated by modules, blocks and filters-->
<script type="text/javascript"
src="http://localhost/moodle2/blocks/head_ins_example/js/main.js"></script>
<script type="text/javascript"
src="http://localhost/moodle2/blocks/head_ins_example/js/teacher.js"></script>
<script type="text/javascript"
src="http://localhost/moodle2/blocks/head_ins_example/js/blockcapability.js"><
/script>
<script type="text/javascript">*<![CDATA[*/*
alert ('This is some inline script. Use inline script for setting global js
vars, etc. The meat of your code should always be in linked script libs. ');
/*]]>*</script>

<!--End of script generated by modules, blocks and filters-->

<!--Start of styles generated by modules, blocks and filters-->
<link rel="stylesheet" type="text/css" href="http://localhost/moodle2/blocks/
head_ins_example/css/main.css" />
<style type="text/css">

h1, h2, h3, h4 {
color:#00a;
}
</style>
```



```
<!--End of styles generated by modules, blocks and filters--><!--<style  
type="text/css">
```