Test

# Use CSS.escape() to escape QuerySelectorAll()

- Link with Extended Character
- Plain Link

I ran into a nasty little bug in Markdown Monster today, where Markdown FootNote links which use `#hash` links on Urls, weren't properly navigating in the HTML Previewer.

Markdown Monster is a Markdown editor and it has a live previewer that lets you see a preview of the generated HTML as you type. The previewer does custom link handling, that forwards the links to the host application which decides how to handle them. Some links are handled by opening new documents, or externally navigating, others are passed along to be processed 'as is' or fixed up to navigate in the same document - such as `#hash` links.

## Hash Handling in the WebView and Chromium

Normally hash handling is totally automatic in the browser - if you add a named hash to the current URL of the page, the page will find the matching `id` in the page and scrolls to it.

Simple right?

**Well, not always.**

MM uses local file paths in the document's current folder to render HTML so that it can correctly manage dependencies like images, scripts, css etc. Because a set of files (a project or just folders) may have a common root that can be specified, MM always uses a `<base>` tag in the generated HTML to point at a root folder. Most of the time that `<base>` path is the same as current document, but in a project setting that folder may be several levels up the hierarchy to point at the project root so paths like `/images/wave.png` can work from down the hierarchy.

The problem with this is that if you have a `<base>` tag in your header, even a simple hash link like this:

```
<a href="#header1">Header 1</a>
```

actually resolves to the base path plus the hash, which is **not the same page as the host page** and results in a bad link display.

The following demonstrates the behavior when:

```
<html>
<head>
 <!-- this messes with #hash nav -->
 <base href="file:///c:/temp/" />
</head>
<body>
 <a href="#header2">header 2</a>

 ...

 <h3 id="header2">Header 2</h3>
</body>
```

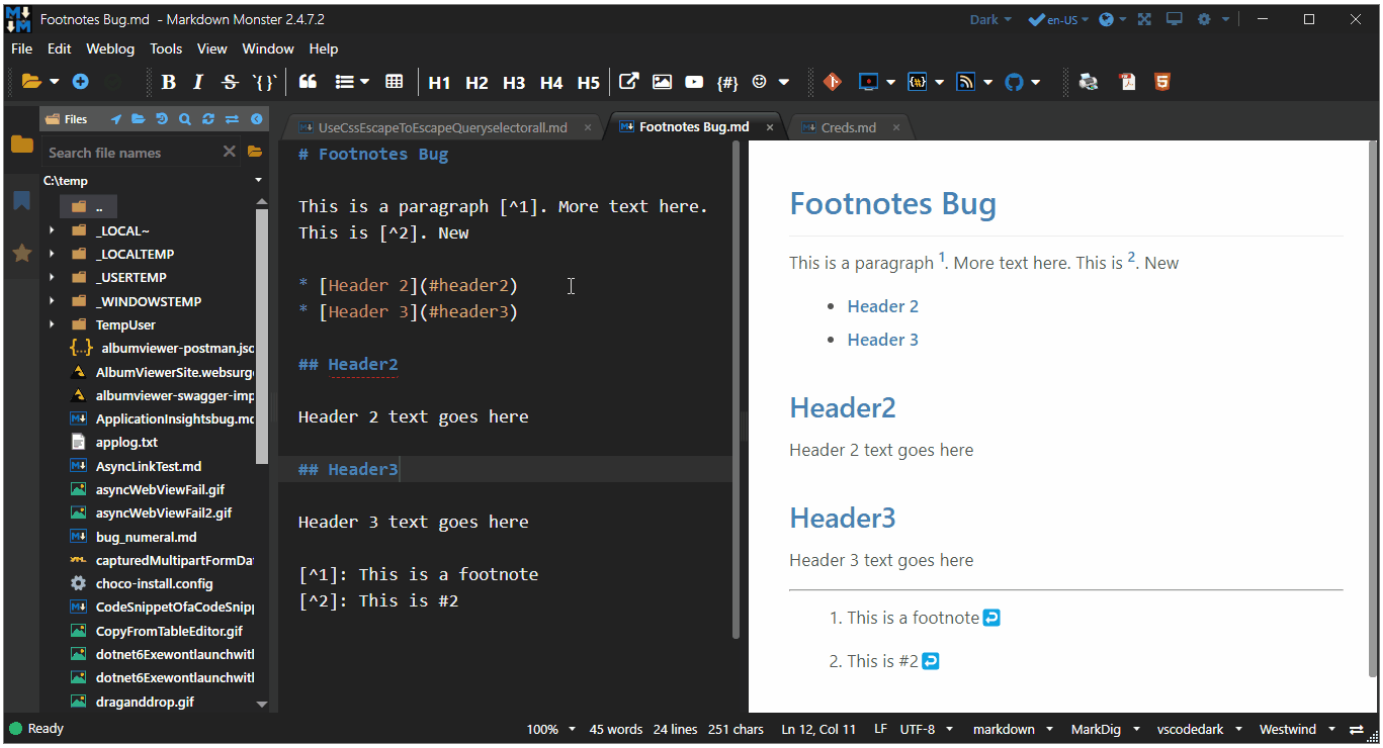
The `#header2` hash now resolves to:

```
file:///c:/temp/#header2
```

which at best is not a valid URL, or at worst navigates to an unrelated document. In the case of the previewer the link fails by opening an OS file view which also is very wrong.

##AD##

Here's what this looks like:




# Growth über Alles

Test

## Explicit Link Handling Required

Because of this MM handles `#hash` links explicitly via a page level link click handler with a section that handles the hashes.

This is my initially - *mostly working* - code:

```
// ... additional logic for external links, docs, and images omitted

// hash = "#header2"
if (hash) {
    var safeHash = decodeURIComponent(hash).substr(1);
    var sel = "a[name='" + safeHash + "'],#" + safeHash;

    var els = document.querySelectorAll(sel);
    if (els && els.length > 0) {
        window.scrollTo(0, els[0].offsetTop - 100);
        return false;
    }

    return true;  // navigate native
}
```

This works better! The above link works just fine when navigating a simple clean hash like `#header2`.

But - remember the beginning of the discussion where I mentioned Markdown Footnotes? Footnote links in Markdown are rendered like this:

```
<!-- footnote references in the text -->
This is footnote <a id="fnref:1" href="#fn:1" class="footnote-ref"><sup>1</sup></a>.

More text here.

This is another footnote <a id="fnref:2" href="#fn:2" class="footnote-ref"><sup>2</sup></a>.


<!-- Footnote definitions at the bottom of document -->
<hr />
<ol>
<li id="fn:1">
<p>This is a footnote<a href="#fnref:1" class="footnote-back-ref">&#8617;</a></p>
</li>
<li id="fn:2">
<p>This is #2<a href="#fnref:2" class="footnote-back-ref">&#8617;</a></p>
</li>
</ol>
```

Notice that there two sets of links that let you jump back and forth between them using `#hash` links.

Using the handler above, the `#header2` links work, but the `#fn:1` links do not.

Experimenting with the JavaScript console I checked `getElementById("fn:1")` and that worked fine, but `querySelectorAll("#fn:1")` would fail.

Test

*What the heck?*

## Selector Encoding - use CSS.Encode()

It took me a bit to realize that the problem wasn't some application logic problem, but rather the fact that the hash contains a `:` which is a **special character for CSS Selectors**. Duh - of course!

The `:` in the `fn:1` Footnote reference is interpreted as a (invalid) **CSS Filter Condition**. A filter condition is something like `a.link:visited` where `:visited` is a filter condition for any visited links. So `fn:1` in `querySelectorAll()` considers `:1` a filter condition, **which of course is invalid**. The selector operation fails, causes an exception and the click handler method then exits without a result value, which in turn causes the default navigation to kin in with the broken `<base>` link. In short, **the navigation now fails**.

To fix this is simple enough: You can use the `CSS.escape()` function to escape a literal string and encodes any Selector specific characters. This includes encoding the `:` character which is escaped with as `:\`.

With that in mind here's the updated code:

```
if (hash) {
    hash = decodeURIComponent(hash).substr(1);

    // THIS
    var safeHash = CSS.escape(hash);  // replaces : with \'

    var sel = "a[name='" + safeHash + "'],#" + safeHash;

    var els = document.querySelectorAll(sel);
    if (els && els.length > 0) {
        window.scrollTo(0, els[0].offsetTop - 100);
        return false;  // handled
    }

    // let browser navigate
}
```

Now all hash link clicks, regardless of special characters work as expected.

The moral of the story is: Remember that querySelectors require escaped values for literals and dynamically provided values that are used in the query, to ensure there's no accidental conflict with a query operator.

`CSS.escape()` is a quick, albeit easy to forget solution.

## Summary

Query Selector encoding is one of those sneaky bugs that you don't anticipate until they bite you in the butt, because things work just fine if you do it one way (`document.gelElementById()`) but not if you do it another (`document.querySelectorAll()`). And that is if you even think about testing for special cases like a provided or literal value that includes a query operator.

BTW, the same rules apply with query selector wrapper libraries like jQuery which also failed with the the raw `#fn:1` tag navigation, for the same reasons.

Test

You can go a long time before you run into a need of `CSS.escape()` - I've certainly **never** had to use it before this particular scenario although I'm certain there are places in much of my Web code where this could become a potential problem.

---

 this post created and published with the **Markdown Monster Editor**