Customisation via Javascript / CSS

Table of Contents

Overview

Code

The following code enables the support for **rendering** additional features, like:

- Highlight lines in code block
- Admonitions / Callouts Github syntax
- Make the top navigation bar stick to the top
- Mermaid
- Table Of Content

```
<!-- Support for highlight lines in code block -->
<style>
  .cm-line-highlight {
    background-color: #fffbcc !important;
    border-left: 3px solid #f0c000;
    padding-left: calc(1em - 3px) !important;
  }
</style>
<script>
(function () {
  const MARKERS = ['', '', ''];


  function highlightLines(root) {
    (root || document).querySelectorAll('.cm-line').forEach(function (line) {
      const text = line.textContent;
      if (!MARKERS.some(m => text.includes(m))) return;
```

```
      line.style.backgroundColor = '#fffbcc';
      line.style.borderLeft = '3px solid #f0c000';
      line.style.paddingLeft = 'calc(1em - 3px)';
      line.style.display = 'block';

      var allSpans = Array.from(line.querySelectorAll('span'));
      var markerChars = '#[hl]';
      var toHide = [];
      var consumed = '';

      for (var i = allSpans.length - 1; i >= 0; i--) {
        var span = allSpans[i];
        if (span.querySelector('span')) continue;
        consumed = span.textContent + consumed;
        toHide.push(span);
        if (consumed.replace(/\s+/g, '') === markerChars) break;
        if (consumed.replace(/\s+/g, '').length > markerChars.length) break;
      }

      if (consumed.replace(/\s+/g, '') === markerChars) {
        toHide.forEach(function (span) {
          span.style.fontSize = '0';
          span.style.letterSpacing = '-1em';
          span.style.opacity = '0';
          span.style.userSelect = 'none';
          span.style.pointerEvents = 'none';
          span.style.display = 'inline-block';
          span.style.overflow = 'hidden';
          span.style.width = '0';
        });
      }
    });
  }

  function debounce(fn, delay) {
    var timer;
    return function () {
      clearTimeout(timer);
      timer = setTimeout(fn, delay);
    };
  }

  function attachToEditor(editor) {
    var content = editor.querySelector('.cm-content');
    if (!content || content._hlObserver) return;

    // Debounced re-apply — waits for CodeMirror to finish its own re-render
    var reapply = debounce(function () {
      highlightLines(editor);
    }, 50);

    var obs = new MutationObserver(reapply);
    obs.observe(content, { childList: true, subtree: true, attributes: true, characterData: true
});
```

```
    content._hlObserver = obs;

    // Also re-apply on any click/focus inside the editor since CM re-renders on interaction
    editor.addEventListener('click', function () {
      setTimeout(function () { highlightLines(editor); }, 0);
      setTimeout(function () { highlightLines(editor); }, 50);
      setTimeout(function () { highlightLines(editor); }, 150);
    });
    editor.addEventListener('focusin', function () {
      setTimeout(function () { highlightLines(editor); }, 0);
      setTimeout(function () { highlightLines(editor); }, 50);
      setTimeout(function () { highlightLines(editor); }, 150);
    });

    highlightLines(editor);
  }

  document.addEventListener('DOMContentLoaded', function () {
    document.querySelectorAll('.cm-editor').forEach(attachToEditor);

    var bodyObserver = new MutationObserver(function (mutations) {
      mutations.forEach(function (mutation) {
        mutation.addedNodes.forEach(function (node) {
          if (node.nodeType !== 1) return;
          if (node.classList && node.classList.contains('cm-editor')) attachToEditor(node);
          node.querySelectorAll && node.querySelectorAll('.cm-editor').forEach(attachToEditor);
        });
      });
    });
    bodyObserver.observe(document.body, { childList: true, subtree: true });

    [200, 500, 1000, 2000].forEach(function (ms) {
      setTimeout(function () {
        document.querySelectorAll('.cm-editor').forEach(attachToEditor);
      }, ms);
    });
  });
})();
</script>

<!-- Admonitions / Callouts Github syntax -->
<style>
  .callout { border-radius: 6px; padding: 12px 16px; margin: 16px 0; border-left: 4px solid; }
  .callout-title { font-weight: 600; margin-bottom: 4px; display: flex; align-items: center; gap:
6px; }
  .callout-note     { background: #dbeafe; border-color: #3b82f6; color: #1e40af; }
  .callout-tip      { background: #dcfce7; border-color: #22c55e; color: #15803d; }
  .callout-important{ background: #ede9fe; border-color: #a855f7; color: #6b21a8; }
  .callout-warning  { background: #fef9c3; border-color: #eab308; color: #854d0e; }
  .callout-caution  { background: #fee2e2; border-color: #ef4444; color: #991b1b; }
  .callout p { margin: 0; }

  /* Hide any icon BookStack injects into or around blockquotes */
  .callout svg,
```

```
  .callout img,
  .callout i,
  .callout span.icon,
  .callout::before,
  .callout::after { display: none !important; }
</style>
<script>
document.addEventListener('DOMContentLoaded', function () {
  const TYPES = {
    NOTE:      { icon: 'ℹ', cls: 'callout-note',      label: 'Note' },
    TIP:       { icon: '', cls: 'callout-tip',       label: 'Tip' },
    IMPORTANT: { icon: '', cls: 'callout-important', label: 'Important' },
    WARNING:   { icon: '⚠', cls: 'callout-warning',   label: 'Warning' },
    CAUTION:   { icon: '', cls: 'callout-caution',   label: 'Caution' },
  };

  document.querySelectorAll('.page-content blockquote').forEach(function (bq) {
    const first = bq.querySelector('p');
    if (!first) return;

    const match = first.textContent.trim().match(/^\[!(NOTE|TIP|IMPORTANT|WARNING|CAUTION)\]/i);
    if (!match) return;

    const type = TYPES[match[1].toUpperCase()];

    // Remove the [!TYPE] line
    const remaining = first.innerHTML.replace(/^\[!(NOTE|TIP|IMPORTANT|WARNING|CAUTION)\]
(<br\s*\/?>)?/i, '').trim();
    if (remaining) first.innerHTML = remaining;
    else first.remove();

    // Build callout div
    const div = document.createElement('div');
    div.className = `callout ${type.cls}`;
    div.innerHTML = `<div class="callout-title">${type.icon} ${type.label}</div>`;
    Array.from(bq.childNodes).forEach(n => div.appendChild(n.cloneNode(true)));

    bq.replaceWith(div);
  });
});
</script>

<!-- Make the top navigation bar stick to the top -->
<style>
  header {
    position: fixed;
    top: 0;
    width: 100%;
    z-index: 1000;
  }

  /* Add padding to the top of the body to prevent content from hiding behind the header */
  body {
    padding-top: 60px; /* Adjust this value based on your header height */
```

```
  }
</style>

<!-- Enable Mermaid -->
<script src="https://cdn.jsdelivr.net/npm/mermaid/dist/mermaid.min.js"></script>
<style>
  .mermaid {
    border-radius: 6px;
    padding: 16px;
  }
</style>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    document.querySelectorAll('code.language-mermaid').forEach(function(el) {
      const div = document.createElement('div');
      div.className = 'mermaid';
      div.textContent = el.textContent;
      el.parentNode.replaceWith(div);
    });

    mermaid.initialize({
      startOnLoad: true,
      theme: 'dark',
    });
  });
</script>

<!-- Display TOC -->
<script>
  document.addEventListener('DOMContentLoaded', function () {
    const content = document.querySelector('.page-content');
    if (!content) return;

    // Find [TOC] placeholder (in a paragraph or standalone)
    const walker = document.createTreeWalker(content, NodeFilter.SHOW_TEXT);
    let tocNode = null;
    while (walker.nextNode()) {
      if (walker.currentNode.textContent.trim() === '[TOC]') {
        tocNode = walker.currentNode;
        break;
      }
    }
    if (!tocNode) return;

    // Gather headings
    const headings = Array.from(content.querySelectorAll('h1, h2, h3, h4'));
    if (!headings.length) return;

    // Ensure headings have IDs
    headings.forEach((h, i) => {
      if (!h.id) h.id = 'toc-heading-' + i;
    });

    // Build ToC
```

```
const nav = document.createElement('nav');
nav.className = 'inline-toc';
nav.innerHTML = '<strong>Table of Contents</strong>';
const ul = document.createElement('ul');

const minLevel = Math.min(...headings.map(h => parseInt(h.tagName[1])));

headings.forEach(h => {
  const level = parseInt(h.tagName[1]) - minLevel;
  const li = document.createElement('li');
  li.style.marginLeft = (level * 16) + 'px';
  const a = document.createElement('a');
  a.href = '#' + h.id;
  a.textContent = h.textContent;
  li.appendChild(a);
  ul.appendChild(li);
});

nav.appendChild(ul);

// Style it
const style = document.createElement('style');
style.textContent = `
  .inline-toc {
    display: inline-block;
    background: var(--color-bg, #f8f9fa);
    border: 1px solid var(--color-border, #dee2e6);
    border-radius: 6px;
    padding: 12px 20px;
    margin: 16px 0;
    font-size: 0.9em;
    min-width: 200px;
    max-width: 400px;
  }
  .inline-toc strong {
    display: block;
    margin-bottom: 8px;
    font-size: 1em;
  }
  .inline-toc ul {
    list-style: none;
    margin: 0;
    padding: 0;
  }
  .inline-toc li {
    margin: 4px 0;
  }
  .inline-toc a {
    color: var(--color-link, #1a73e8);
    text-decoration: none;
  }
  .inline-toc a:hover {
    text-decoration: underline;
  }
```

```
    `;
    document.head.appendChild(style);

    // Replace [TOC] node with the nav
    tocNode.parentNode.replaceWith(nav);
  });
</script>
```

Revision #1
Created 2026-03-14 01:34:15 UTC by Dario
Updated 2026-03-14 01:39:49 UTC by Dario