MicroCODE Software Engineering Services
Project Coding Standards

MCX-S03 (Internal HTML-CSS Style Guide) v003.docx

Development Environment: Microsoft **Visual Studio Code**
OS: HTML5 Browser Environment
Platform: Open Source **ES6, HTML5, CSS3**
Language: JavaScript (**ECMAScript 6** – ES6)
Tools: JSLint, JSHint

MicroCODE HTML-CSS Style Guide

This was adopted from the **MIT xPRO CSS Style Guide**, differences are noted.
The MIT guide was in turn adapted from the Airbnb style guide.

Table of Contents

Standard Terminology (MicroCODE)

Any Style Guide should start with **coding terminology** because there is a general lack of standardized terms.

( ) – these are **PARENs**, short for parentheses, they are not 'round brackets' or 'round braces', parens by our definition are a pair of round delimiters.

[ ] – these are **BRACKETs**. They are not braces, they are not 'square brackets', brackets are by our definition are a pair of square delimiters.

{ } – these are **BRACEs**. They are not brackets, they are not 'curly brackets' nor 'curly braces', braces are by our definition are a pair of a 'curly' delimiters. Saying 'curly braces' is redundant, saying 'curly brackets' is just wrong.

< > – these are **ANGLEs**. They are not brackets, they are not 'angled brackets' nor 'angled braces', angles are by our definition a pair of angular delimiters.

In typography all of these are referred to generically as 'brackets'. A nice article explaining this: **https://type.today/en/journal/brackets**. But we are not talking about typesetting, we are talking about coding, and precise language eliminates confusion and mistakes; it produces consistent results and saves time & money. And so, for all our code and documentation…

Code Delimiters

```
( ) = PARENS:   parameter grouping, operand precedence, quantities
[ ] = BRACKETS: indexing, array formation
{ } = BRACES:   code blocks, initialization values (compiler)
< > = ANGLES:   substitution identifier, option grouping
```

Indents, Braces, Blocks, and Aligned-Code (MicroCODE)

Common JavaScript Style Guides like to use Kernighan & Richie (K&R) '**Egyptian Brackets**'.

Right off the bat, they are really not talking about "brackets" they are talking about "braces".
Our opinion is that this style 'saves' one line break per block of code at the expense of readability.
Why? Because the key word expressions like 'if' and 'else' overlap and offset the code of the clause.
This style saved screen space when people coded on 24-Line CRTs (like the VT52s, VT100s, VT220s, etc.) and saved paper when people printed code. We have neither of these limitations in the 21st Century; white space is your friend, delimiter alignment is your friend, language keywork alignment is your friend.

Note: Aligned-Code is also known as **Allman Style** (after Eric Allman) or **BSD Style** – Berkeley Software Distribution.

```
if (condition) {
    // true code
} else {
    // false code
}

try {
    // protected code
} catch {
    // correction code
} finally {
    // exit code
}
```

K&R Style

The above is K&R, below is BSD which is our preferred JavaScript Style (and C/C++/C# Style as well).
The braces are aligned.
The key words are aligned.
All code blocks are held within aligned braces.
All conditional code has natural white space around it via the braces.
People will argue that K&R is easy to read, but they are relaying on colorized text editors that are hiding the readability issue in the bare text. **There is no world in which K&R is easier to read and maintain than BSD.**

```
if (condition)
{
    // true code
}
else
{
    // false code
}

try
{
    // protected code
}
catch
{
    // correction code
}
finally
{
    // exit code
}
```

BSD Style

Notice in the second example the conditional code is naturally separated by white space created by the balanced braces, this makes the code far more readable that the 'Egyptian Brackets'. And there are no excuses based on typing speed, as these preferences can all be enforced automatically by VS CODE Settings.

White space is your friend and aids in readability; half the effort of maintaining code is readability and consistency.

As an extension, when building conditional execution, braces should always be used—and all code blocks should be placed on the lines following the conditional expression.

```
if (condition)  …  // true code - DO NOT DO THIS

if (condition)
      …  // true code - DO NOT DO THIS

if (condition)
{
    … // true code - ALWAYS DO THIS, conditional code in a BLOCK of BRACEs, 'ALIGNED'
    … // here is example a line added in the future, no reformatting required
}
```

This first reason for this is consistency and long-term maintenance. If the first two examples ever need editing—where an additional line of code needs to be added to the execution clause—the first thing that must be done is the addition of braces and the reformatting of the lines… a burden placed on the future coder by the original author. If line(s) of code are added in a hurry—and no braces are added—you just cost the next coder (or your future self) a good :10 minutes figuring out why their condition code is not working. It just not worth it.

The second reason is readability, it's just far easier to read all conditional code the same way, everywhere in a project, if it is all formatted exactly the same way.

It is often said: "Cleanliness is Next to Godliness", we believe "Consistency is Next to Godliness".

The universe works on a set of consistent rules for a reason. If you code consistently—always following the same rules, always following the same patterns, where all code looks like it came from the same author, where all Classes follow the same format—you and your Team will reap the benefits now and forever. If your code is utterly consistent your Team members can code with assumptions that will always be true… **that** will speed up development more than any style can by saving typing white space or braces.

See **Appendix A: MicroCODE JS Class Structure** as an example of using a Template to make following these rules simple.

"MCODE: Code like a Machine – Consistently, Simply, Explicitly, and for Readability." SM

The combination of BSD Style, with the MCODE Rules, like always starting new modules from approved templates—no matter how simple the new Class or Module may be—we call…

MCODE Style

See our JavaScript Style Guide for complete details on JS code standards:

git@github.com:MicroCODEIncorporated/JavaScriptSG.git

CSS Terminology

Rule declaration

A “rule declaration” is the name given to a selector (or a group of selectors) with an accompanying group of properties.

Here's an example:

```
.listing
{
  font-size: 18px;
  line-height: 1.2;
}
```

Selectors

In a rule declaration, “selectors” are the bits that determine which elements in the DOM tree will be styled by the defined properties. Selectors can match HTML elements, as well as an element's class, id, or any of its attributes.

Here are some examples of selectors:

```
h1
{
  /*...*/
}

.my-elements-class
{
  /* ... */
}
```

Properties

Finally, properties are what give the selected elements of a rule declaration their style. Properties are key-value pairs, and a rule declaration can contain one or more property declarations.

Property declarations look like this:

```
/* some selector */
{
  background: #f1f1f1;
  color: #333;
}
```

CSS Formatting

Formatting

A “rule declaration” is the name given to a selector (or a group of selectors) with an accompanying group of properties.

- Use soft tabs (2 spaces) for indentation.
- Prefer dashes over camelCasing in class names. (some-class vs. someClass)
- Put a space before the opening brace **{** in rule declarations.
- Don't use ID selectors.
- In properties, put a space after, but not before, the **:** character.
- Put closing braces **}** of rule declarations on a new line.
- Put blank lines between rule declarations.

Bad

```
.avatar{
    border-radius:50%;
    border:2px solid white; }
.no, .nope, .not_good {
    // ...
}
#lol-no {
  // ...
}
```

Good

```
.avatar
{
  border-radius: 50%;
  border: 2px solid white;
}

.one,
.selector,
.per-line
{
  // ...
}
```

A Quick Note on Dashes: Using dashes (not-camel-case) vs. other casing, aka camelCase is a hotly debated preference, however, it does help with browser implementations to use dashes. For more, check out this Stack Overflow thread.

Comments

Comments in CSS are written inside **/* */** marks.

For long comments, put the comments on their own lines, between a beginning and end **/* */** symbol.

Bad

```
.container {
  font-size: 10em; /*a very long, exhaustive comment here with multiple points that should
really be on its own line(s)*/
}
```

Good

```
.container
{
    font-size: 10em; /* short comment */
}

.container
{
  /*
    a long,
    multiline comment
    looks like this - avoid doing this!
  */
  font-size: 10em;
}

.container
{
  /* for longer comments preface the line being referenced - better */
  font-size: 10em;
}
```

ID Sectors

While it is possible to select elements by ID in CSS, it should generally be considered an anti-pattern. ID Selectors introduce an unnecessarily high level of specificity to your rule declarations, and they are not reusable.

For more on this subject, read **CSS Wizardry's article** on dealing with specificity.

A Note on Functionality

While if you don't follow these style guide's rules, your CSS will still work, it can cause a lot of confusion among yourself and your team members if your code is all formatted differently. It is highly recommended to stick to a style guide so that development can happen in a clean and fast manner.

Appendix A: MicroCODE HTML-CSS Structure

Our template establishes a standard ordering of the HTML file elements as shown below. This is the ordering enforced by the MicroCODE HTML-CSS Template. (Code Explicitly, Code for Readability, Code for Readability).

Within a **page** (`mcodeTemplate.html`) groups **elements** in this order:

- CSS Style Definitions (`<style>`)
- HTML Header (`<header>`)
- HTML Body (`<body>`)
- External Data Sources (`<script src=>`)
- External JavaScript Code (`<script src=>`)
- Page JavaScript (`<script>`)
  - Private Constants (`const CONSTANT_NAME`)
  - Private Fields (`let _privateName`)
  - Private Functions (`function functionName(param)`)
  - Callback Functions (`function functionName(param)`)
  - Event Handlers (`function functionName(param)`)
  - Default Execution (`JS Code`)

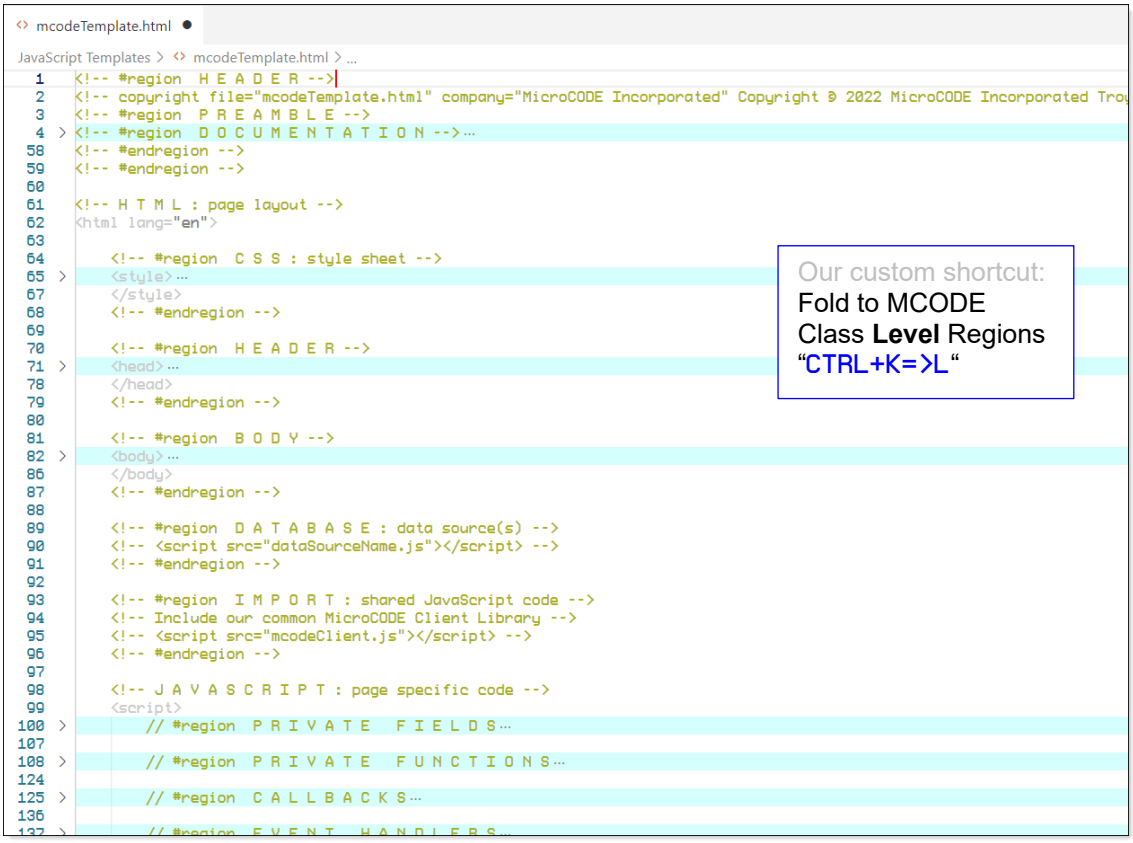
HTML-CSS Template (MicroCODE)

`mcodeTemplate.html` provides a standard template for an HTML page with CSS and JavaScript included.

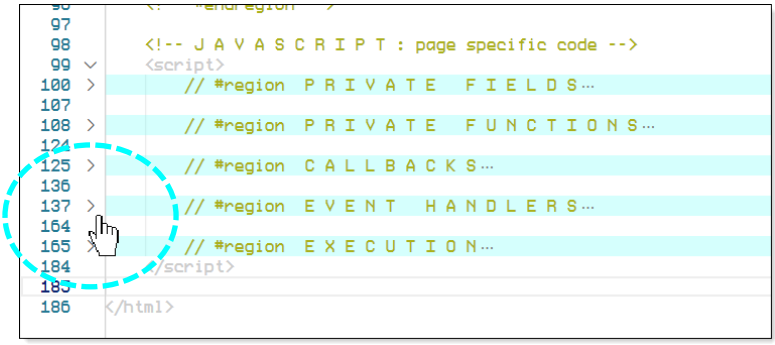
This template is built with **Code Folding** support through use of…

"`<!-- #region -->`" and
"`<!-- #endregion -->`".

This will seem like a lot of unnecessary 'syntactic sugar'—I hate that phrase BTW—reading through the following and understanding what it enables in VS CODE should make the value clear. Start with the entire file 'folded' to Class level. "`CTRL+K=>3`".



Then 'unfold' the area you are working my clicking the "`>`" for that Region.



This leaves everything you are not working on ‘hidden’ via the ‘folded code’ support in VS Code.

```
98      <!-- J A V A S C R I P T : page specific code -->
99      <script>
100 >       // #region  P R I V A T E   F I E L D S ···
107
108 >       // #region  P R I V A T E   F U N C T I O N S ···
124
125 >       // #region  C A L L B A C K S ···
136
137         // #region  E V E N T   H A N D L E R S
138         /*
139          * btnUI_*_Click() - event handlers for UI buttons.
140          */
141
142         // Button #1 handler
143         function btnUI_Button1_Click(messageData1, callBack)
144         {
145             // Handle 'Click'...
146             console.log(callBack(messageData1));
147         }
148
149         // Button #2 handler
150         function btnUI_Button2_Click(messageData2, callBack)
151         {
152             // Handle 'Click'...
153             console.log(callBack(messageData2));
```

All the ‘code folding’ can be opened with… “CTRL+K=>J“.

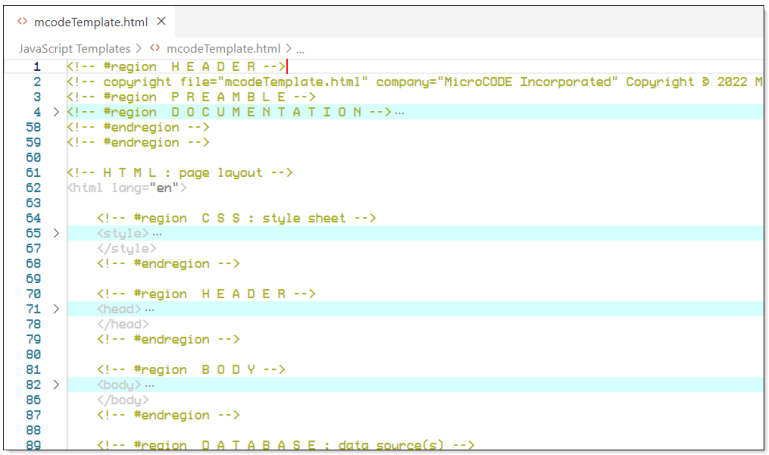
mcodeTemplate.html

JavaScript Templates > mcodeTemplate.html > html

```
1   <!-- #region  H E A D E R -->
2   <!-- copyright file="mcodeTemplate.html" company="MicroCODE Incorporated" Copyright © 2022 Micro
3   <!-- #region  P R E A M B L E -->
4   <!-- #region  D O C U M E N T A T I O N -->
5   <!--
6    *      Title:    MicroCODE Common HTML Template
7    *      Module:   Modules (MicroCODE:mcodeTemplate.html)
8    *      Project:  MicroCODE Common Web Pages
9    *      Customer: Internal
10   *      Creator:  MicroCODE Incorporated
11   *      Date:     March 2022
12   *      Author:   Timothy J McGuire
13   *
14   *      Designed and Coded: 2022 MicroCODE Incorporated
15   *
16   *      This software and related materials are the property of
17   *      MicroCODE Incorporated and contain confidential and proprietary
18   *      information. This software and related materials shall not be
19   *      duplicated, disclosed to others, or used in any way without the
20   *      written of MicroCODE Incorporated.
21   *
22   *
23   *      DESCRIPTION:
24   *      ------------
25   *
26   *      This module implements the MicroCODE's Common HTML Template.
27   *      This file is copied to start all MicroCODE Web Pages.
28   *
29   *
30   *      REFERENCES:
31   *      -----------
32   *
33   *      1. MicroCODE JavaScript Style Guide
34   *         Local File: MCX-S02 (Internal JS Style Guide).docx
35   *         https://github.com/MicroCODEIncorporated/JavaScriptSG
36   *
37   *      2. ...
38   *
39   *
```

Standard shortcut:
Unfold ALL Regions
“CTRL+K=>J“

Everything can be ‘folded’ to LEVEL 0 with… “CTRL+K=>0“.



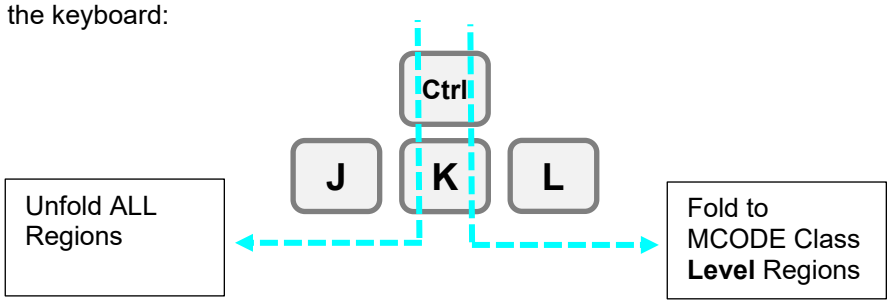
And you can restore your view of the overall HTML/CSS structure (locked by **MCODE Style**) at any time with…

“CTRL+K=>J“ “CTRL+K=>3“.

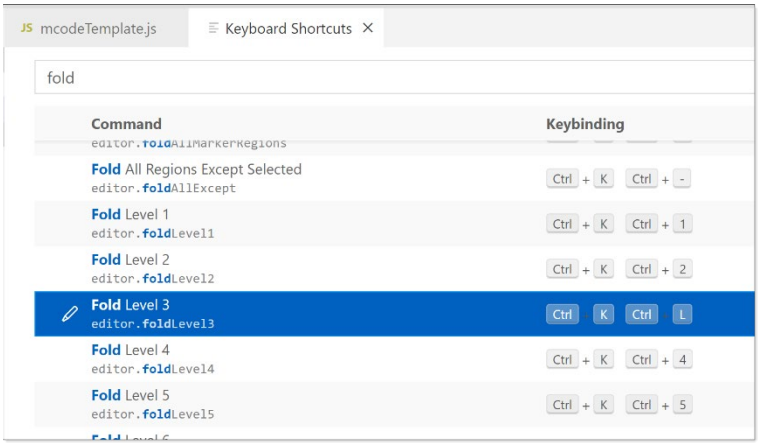
NOTE: You cannot get the same view with “CTRL+K=>3“ without starting with “CTRL+K=>J“. These commands are ‘relative’ to the current view, not ‘absolute’… i.e.: “CTRL+K=>3“ will not always give you the same result by itself.

I use this all day every day when navigating files, so I have remapped “CTRL+K=>L“ to equal “CTRL+K=>3“.

So, on the keyboard:



- File → Preferences → Keyboard Shortcuts



```
<!-- #region  H E A D E R -->
<!-- copyright file="mcodeTemplate.html" company="MicroCODE Incorporated" Copyright © 2022 MicroCODE Incorporated Troy, MI
author="Timothy J. McGuire" -->
<!-- #region  P R E A M B L E -->
<!-- #region  D O C U M E N T A T I O N -->
<!--
 *      Title:    MicroCODE Common HTML Template
 *      Module:   Modules (MicroCODE:mcodeTemplate.html)
 *      Project:  MicroCODE Common Web Pages
 *      Customer: Internal
 *      Creator:  MicroCODE Incorporated
 *      Date:     March 2022
 *      Author:   Timothy J McGuire
 *
 *      Designed and Coded: 2022 MicroCODE Incorporated
 *
 *      This software and related materials are the property of
 *      MicroCODE Incorporated and contain confidential and proprietary
 *      information. This software and related materials shall not be
 *      duplicated, disclosed to others, or used in any way without the
 *      written of MicroCODE Incorported.
 *
 *
 *      DESCRIPTION:
 *      ------------
 *
 *      This module implements the MicroCODE's Common HTML Template.
 *      This file is copied to start all MicroCODE Web Pages.
 *
 *
 *      REFERENCES:
 *      -----------
 *
 *      1. MicroCODE JavaScript Style Guide
 *         Local File: MCX-S02 (Internal JS Style Guide).docx
 *         https://github.com/MicroCODEIncorporated/JavaScriptSG
 *
 *      2. ...
 *
 *
 *      DEMONSTRATION VIDEOS:
 *      ---------------------
 *
 *      1. ...
 *
 *
 *
 *      MODIFICATIONS:
 *      --------------
 *
 *  Date:         By-Group:   Rev:     Description:
 *
 *  04-Mar-2022   TJM-MCODE  {0001}    New file for starting all HTML files to maintain common structure.
 *
 *
-->

<!-- #endregion -->
<!-- #endregion -->
<!-- #endregion -->

<!-- H T M L : page layout -->
<html lang="en">

    <!-- #region  C S S : style sheet -->
    <style>

    </style>
    <!-- #endregion -->

    <!-- #region  H E A D E R -->
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible"
              content="IE=edge">
        <meta name="viewport"
              content="width=120, initial-scale=1.0">
        <title>MIT xPRO: WEEK n</title>
    </head>
    <!-- #endregion -->

    <!-- #region  B O D Y -->
    <body>
        <button id="btnUI_Button1">Button 1</button>
        <button id="btnUI_Button2">Button 2</button>
        <button id="btnUI_Button3">Button 3</button>
    </body>
    <!-- #endregion -->
```

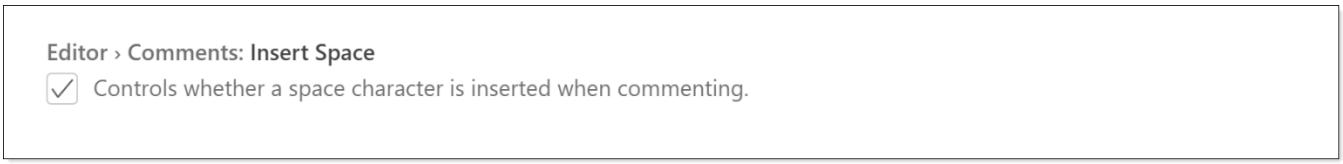
```
<!-- #region  D A T A B A S E : data source(s) -->
<!-- <script src="dataSourceName.js"></script> -->
<!-- #endregion -->

<!-- #region  I M P O R T : shared JavaScript code -->
<!-- Include our common MicroCODE Client Library -->
<!-- <script src="mcodeClient.js"></script> -->
<!-- #endregion -->

<!-- J A V A S C R I P T : page specific code -->
<script>
    // #region  P R I V A T E  C O N S T A N T S

    const MIN_VALUE = 1;
    const MAX_VALUE = 999;
    const CLASS_TYPE = 'Example';

    // #endregion


    // #region  P R I V A T E  F I E L D S

    let _notImplemented = ' is not implemented.';
    let _privateField1 = 0;
    let _privateField2 = [];

    // #endregion

    // #region  P R I V A T E  F U N C T I O N S
    /**
     * @function functionName() -- function description.
     *
     * @param {string} param1 parameter #1 description.
     * @param {number} param2 parameter #2 description.
     * @returns {object} description of return value.
     */
    var functionName = function (param1, param2)
    {
        let value = null;

        return value;
    };

    // #endregion

    // #region  C A L L B A C K S
    /*
     * callback*() - callback* description.
     */
    function callbackFormatter(messageData)
    {
        // handle item manipulation...
        return messageData + _notImplemented;
    }

    // #endregion

    // #region  E V E N T  H A N D L E R S
    /*
     * btnUI_*_Click() - event handlers for UI buttons.
     */

    // Button #1 handler
    function btnUI_Button1_Click(messageData1, callBack)
    {
        // Handle 'Click'...
        console.log(callBack(messageData1));
    }

    // Button #2 handler
    function btnUI_Button2_Click(messageData2, callBack)
    {
        // Handle 'Click'...
        console.log(callBack(messageData2));
    }

    // Button #3 handler
    function btnUI_Button3_Click(messageData3, callBack)
    {
        // Handle 'Click'...
        console.log(callBack(messageData3));
    }

    // #endregion

    // #region  E X E C U T I O N

    // Load data sources...
```

```
        // Map Event Handlers...
        document.getElementById("btnUI_Button1")
            .addEventListener("click", () => { btnUI_Button1_Click('Button #1', callbackFormatter); });

        document.getElementById("btnUI_Button2")
            .addEventListener("click", () => { btnUI_Button2_Click('Button #2', callbackFormatter); });

        document.getElementById("btnUI_Button3")
            .addEventListener("click", () => { btnUI_Button3_Click('Button #3', callbackFormatter); });

        // Set Timeout Events...

        // Run Timed Events...

        // #endregion
    </script>

</html>
```

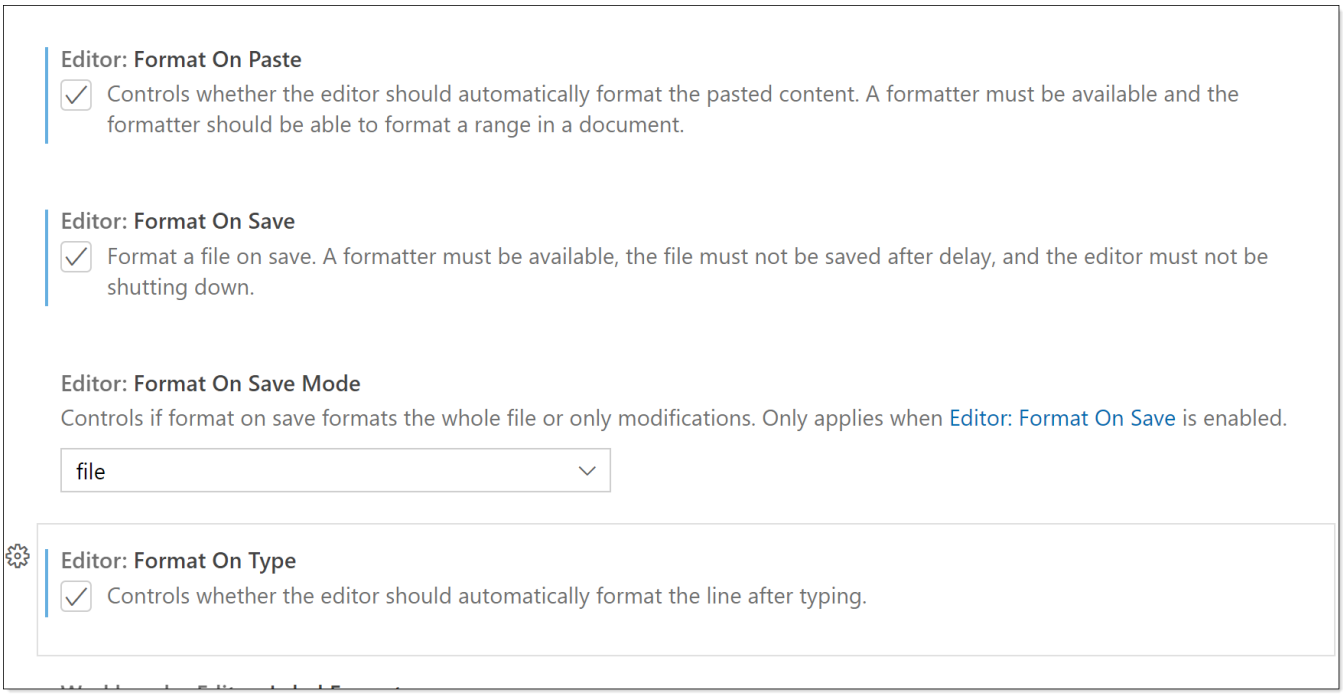
Appendix B: Visual Studio Code Settings

These are the VS Code settings recommended to maintain **MCODE Style** automatically.

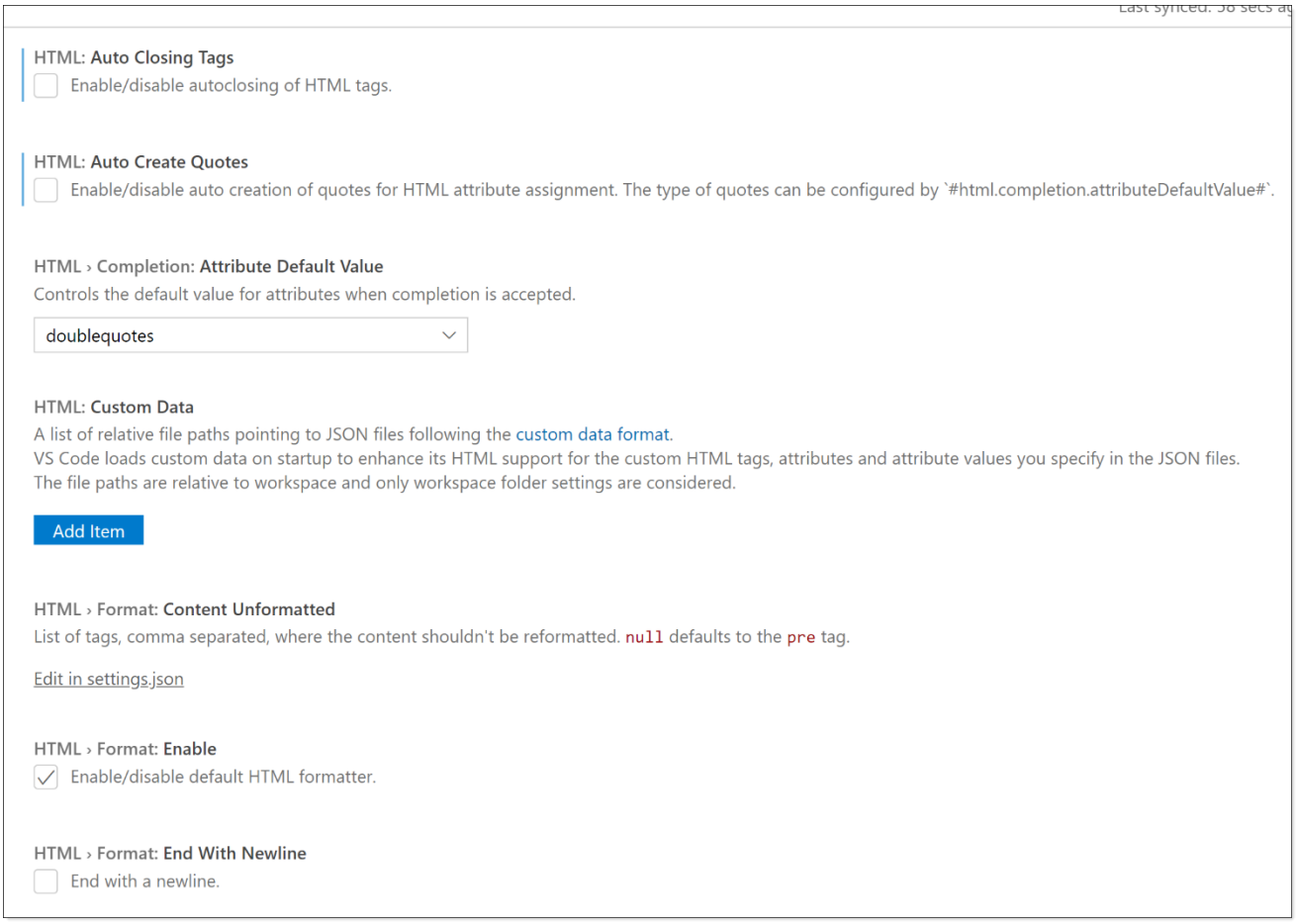
- Automatic Spaces



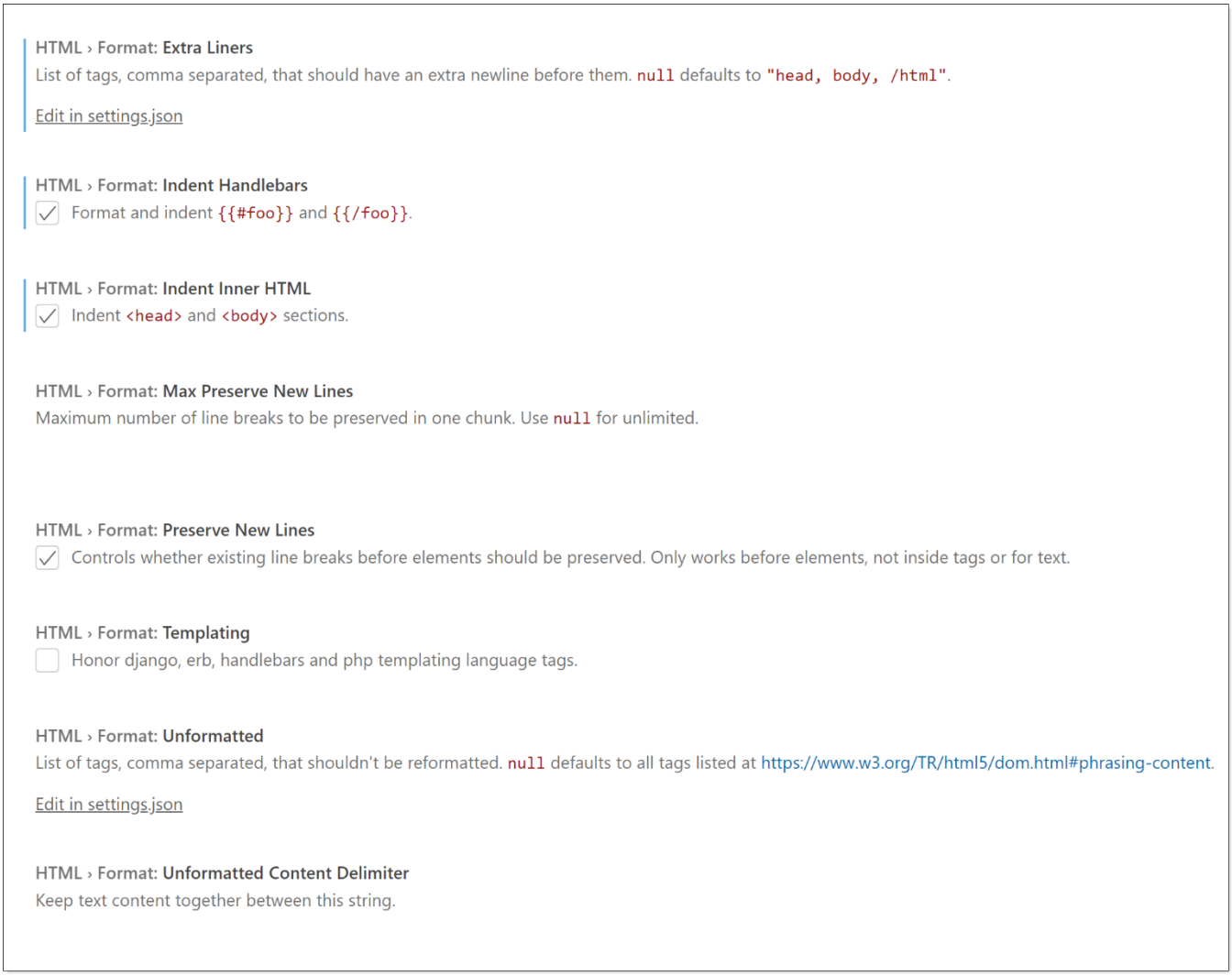
- When to auto-format



- HTML auto-formatting options



- HTML auto-formatting options (continued)



HTML › Format: **Extra Liners**
List of tags, comma separated, that should have an extra newline before them. `null` defaults to `"head, body, /html"`.

Edit in settings.json

HTML › Format: **Indent Handlebars**
☑ Format and indent `{{#foo}}` and `{{/foo}}`.

HTML › Format: **Indent Inner HTML**
☑ Indent `<head>` and `<body>` sections.

HTML › Format: **Max Preserve New Lines**
Maximum number of line breaks to be preserved in one chunk. Use `null` for unlimited.

HTML › Format: **Preserve New Lines**
☑ Controls whether existing line breaks before elements should be preserved. Only works before elements, not inside tags or for text.

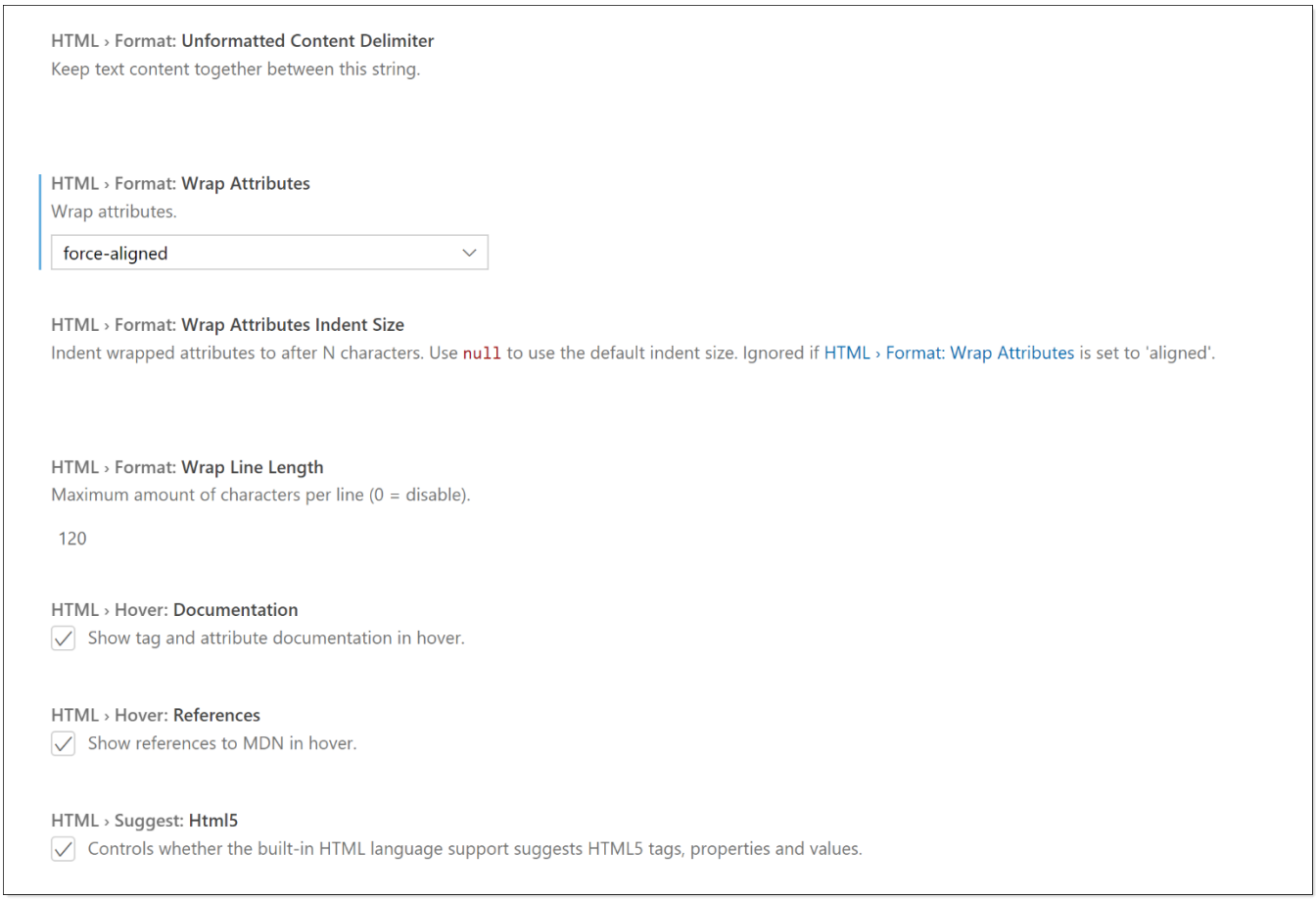
HTML › Format: **Templating**
☐ Honor django, erb, handlebars and php templating language tags.

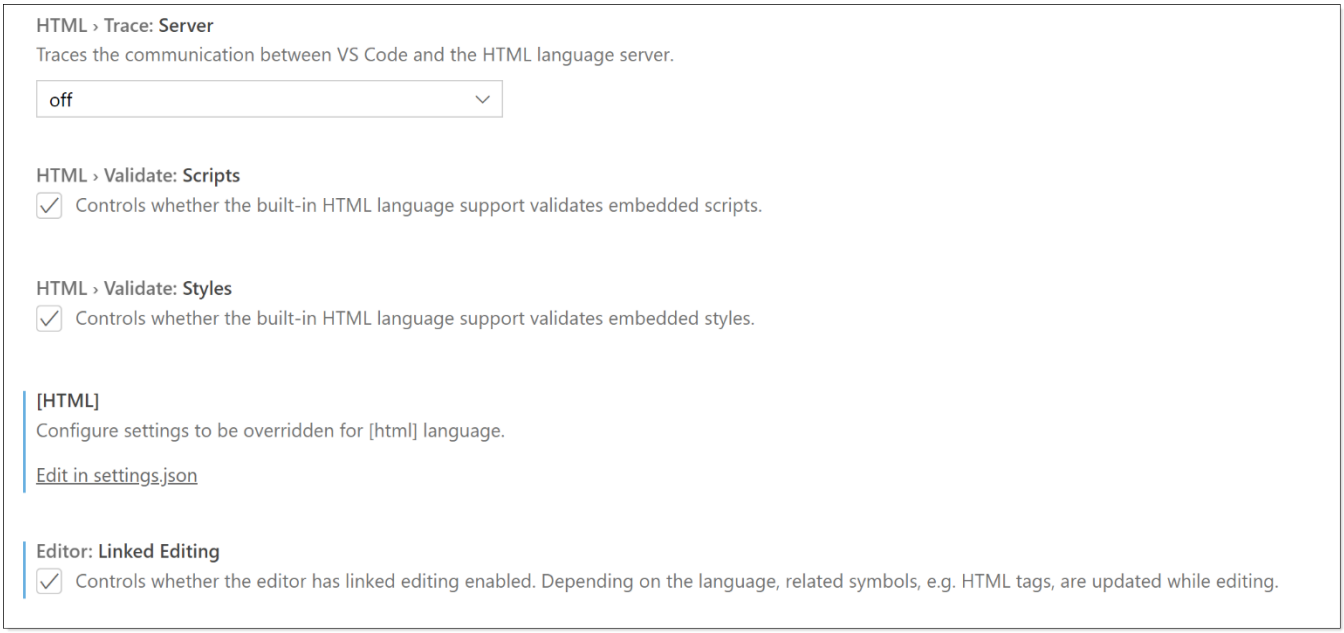
HTML › Format: **Unformatted**
List of tags, comma separated, that shouldn't be reformatted. `null` defaults to all tags listed at https://www.w3.org/TR/html5/dom.html#phrasing-content.

Edit in settings.json

HTML › Format: **Unformatted Content Delimiter**
Keep text content together between this string.

- HTML auto-formatting options (continued)





Licenses

Original Airbnb CSS Style Guide

MIT License

Copyright © 2015 Airbnb

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

MicroCODE HTML-CSS Style Guide

MIT License

Copyright © 2022 MicroCODE, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.