Element Selectors

Element selectors within the Monetate Builder tools support both CSS3 and Sizzle selector syntax.

You can use attribute selectors and pseudo-class selectors in both stylesheets and the **Element Selector** field. You can only use Sizzle selectors in the **Element Selector** field.

The expandable sections below contain information and examples about the types of selectors that the Monetate platform supports.

Selector Types

You can use any element (also known as a node) on a page as a selector to target element(s) of that node type. You can target any number of elements in HTML, such as `<div>`, `<img>`, or `<p>`. You can also directly select any HTML5 nodes as well. This includes `<aside>`, `<nav>`, and many others. You can review a full list of HTML elements at MDN Web Docs.
The following example selects all links, or `<a>` elements, and then applies CSS styling to those elements.

```
a {
  color: blue;
}
```

You can target multiple elements at the same time by formatting with comma separation. The following example selects all paragraphs (`<p>`) and all spans (`<span>`) and then applies CSS styling to those elements.

```
p, span {
  color: #000;
}
```

You can also write a selector to target every type of element on the page. This is useful for *global resets* of all elements, as demonstrated in the following example:

```
* {
  margin: 0;
  padding: 0;
}
```

Attribute Selectors

The attribute selector is used to select elements with a specified attribute. The following example selects all `<a>` elements with a target attribute and applies CSS styling to those elements:

```
a[target] {
  background-color: yellow;
}
```

An attribute selector isn't limited to the existence of the attribute on the element. You can also specify the attribute's value.

ID

You can write some specific attributes using abbreviations in CSS and Sizzle. An ID is a unique identifier attribute that should only ever be present on one element per page and is formatted `#id`.
This example applies a border only to the one element with this ID.
`<a href="#" id="logoLink">Test</a>`

```
#logoLink {
  border: 1px solid red;
}
```

Class

You can write some specific attributes using abbreviations in CSS and Sizzle. A class is an identifier attribute that may be present on one or more elements per page and is formatted `.class`.
This example applies a border to only the elements with the class `blue`.
`<a href="#" class="blue">Test</a>`

```
.blue {
  color: blue;
}
```

Attribute Value Equals String

The following example selects all `<a>` elements with a target attribute that has a value *exactly equal to the string* `valueOne`. This is case-sensitive and applies CSS styling to those elements.
`<a href="#" target="valueOne">Test</a>`

```
a[target="valueOne"] {
  background-color: yellow;
}
```

Attribute Value Equals Word

The following example selects all `<a>` elements with a target attribute that has a value *exactly equal to the word* `valueOne`. This is case-sensitive and applies CSS styling to those elements.
`<a href="#" target="valueOne">Test</a>`

```
a[target=valueOne] {
  background-color: yellow;
}
```

Attribute Value Contains String

The following example selects all `<a>` elements with a target attribute that has a value *containing the string* `One`. This is case-sensitive and applies CSS styling to those elements.
`<a href="#" target="valueOne">Test</a>`

```
a[target*="One"] {
  background-color: yellow;
}
```

Attribute Value Contains Word

The following example selects all `<a>` elements with a `data` attribute that has a value *containing the whole word* `one`. This is case-sensitive and applies CSS styling to those elements.

```
<a href="#" data="value one">Test</a>
```

```
a[data~=one] {
  background-color: yellow;
}
```

Attribute Value Starts with String

The following example selects all `<a>` elements with a target attribute that has a value *starting with the string* `val`. This is case-sensitive and applies CSS styling to those elements.

```
a[target^="val"] {
  background-color: yellow;
}
```

Attribute Value Starts with Word

The following example selects all `<a>` elements with a target attribute that has a value *starting with the word* `value`. This is case-sensitive and applies CSS styling to those elements. The value must be a whole word that is either alone, such as `class="value"`, or that is followed by a hyphen (-), such as `class="value-three"`.

```
<a href="#" data="value-one">Test</a>
```

```
a[data|="value"] {
  background: yellow;
}
```

Attribute Value Ends with

The following example selects all `<a>` elements with a target attribute that has a value *ending with the string* `ueOne`. This is case-sensitive and applies CSS styling to those elements.

```
<a href="#" data="value-one">Test</a>
```

```
a[data$="ue-one"] {
  background-color: yellow;
}
```

You can also write the selector without quotes:

```
a[data$=one] {
  background-color: yellow;
}
```

Structural Selectors

Direct Descendant

Use the `>` selector to select elements that are *exactly one level* down in the markup structure and no deeper. The following example selects all `div` elements *only* one level down in the markup of the `.main` parent element and applies CSS styling to those elements. It does not apply CSS styling to any others that may be two or more levels farther down in the markup.

```
.main > div {
  background-color: yellow;
}>
```

Adjacent Sibling Combinator

Use the `+` selector to select one or more elements *directly after* another specific element. The following example selects all `div` elements that directly follow another `div` within the `.main` parent element.

```
.main > div + div {
  background-color: yellow;
}
```

General Sibling Combinator

Use the `~` selector to select one or more elements that are after another element. The element that you select doesn't need to immediately succeed the first element but can appear anywhere after it. The following example selects all `div` elements that generally follow another `div` within the `.main` parent element.

```
.main > div ~ div {
  background-color: yellow;
}
```

Structural Pseudo-Class Selectors

Pseudo-class selectors define a special state of an element. The syntax is as follows:

```
selector:pseudo-class {
  property: value;
}
```

You may be familiar with CSS pseudo-class selectors such as `:link`, `:visited`, `:hover`, and `:focus`. These selectors all allow for styling on various states of elements.

:first-child

The `:first-child` pseudo-class represents any element that is the first child element of its parent element. In the following example, the selector applies CSS styling to every `<p>`, which is the first child element within an element with the `main` class.

```
.main p:first-child {
  background-color: yellow;
}
```

:last-child

The `:last-child` pseudo-class represents any element that is the last child element of its parent element. In the following example, the selector applies CSS styling to every `<p>`, which is the last child element within an element with the `main` class.

```
.main p:last-child {
  background-color: yellow;
}
```

:first-of-type

The `:first-of-type` pseudo-class represents any element that is the last child element of its parent element *of a specific type*.
In the following example, the selector applies CSS styling to only `<p>` elements with the `blueText` class, which is the first child element within an element with the `main` class.

```
.main p.blueText:first-of-type {
background-color: yellow;
}
```

The notable difference with this selector is that the *type* matters. Since `<p>` is specified, the Personalization platform does not consider other elements, such as `<span>` or `<div>` that might also have the `blueText` class.

:last-of-type

The `:last-of-type` pseudo-class represents any element that is the last child element of its parent element *of a specific type*.
In the following example, the selector applies CSS styling to only `<p>` elements with the `blueText` class, which are the last child element within an element with the `main` class.

```
.main p.blueText:last-of-type {
  background-color: yellow;
}
```

The notable difference with this selector is that the *type* matters. Since `<p>` is specified, the Personalization platform doesn't consider other elements, such as `<span>` or `<div>`, that also have the `blueText` class.

:only-of-type

The `:only-of-type` pseudo-class matches every element that is the only child of its type of its parent.
In the following example, the selector applies CSS styling to the only `<p>` element, which is a child element within an element with the `main` class.

```
.main p:only-of-type {
  background-color: yellow;
}
```

This selector doesn't work if there are multiple `<p>` elements present within a parent. It only applies to instances in which there is a single element type within a parent.

:nth-child

The `:nth-child` pseudo-class selector matches every element that is the *n*th child, regardless of type, of its parent where *n* can be a number, a keyword, or a formula.

Number

In the following example, the selector applies CSS styling to every `<p>` element, which is the *second child* element of an element with the `main` class.

```
.main p:nth-child(2) {
  background-color: yellow;
}
```

Odd or Even

In the following example, the selector applies CSS styling to every `<p>` element whose index is odd or even within an element with the `main` class.

```
.main p:nth-child(odd) {
  background-color: yellow;
}

.main p:nth-child(even) {
  background-color: blue;
}
```

Formula

`:nth-child` can be used with a formula *(an + b)*. `a` represents a cycle size, `n` is a counter, which starts at 0, and `b` is an offset value that you can use if, for example, you want to begin counting after the fourth item.
In the following example, the selector applies CSS styling to every `<p>` element whose index is a multiple of 3 within an element with the `main` class.

```
.main p:nth-child(3n+0) {
  background-color: yellow;
}
```

It is not necessary to include an offset value in the formula. If you do not, it is implied to be `0`. You can achieve the same effect with the following example:

```
.main p:nth-child(3n) {
  background-color: yellow;
}
```

The formulas within `:nth-child` can be confusing and complex. The blog CSS-Tricks published How nth-child Works that can help you brush up on using `:nth-child` effectively.

:nth-of-type

`:nth-of-type` works like `:nth-child`, and you can use it with a formula *(an + b)*. `a` represents a cycle size, `n` is a counter, which starts at 0, and `b` is an offset value. The notable difference is that the *element type* matters in the calculation of the formula.
In the following example, the selector applies CSS styling to every `<p>` element with a class of `blueText` whose index is a multiple of 3 within an element with the `main` class.

```
.main p.blueText:nth-of-type(3n) {
  background-color: yellow;
}
```

Since `<p>` is specified, other elements, such as `<span>` or `<div>`, that might also have the `blueText` class are not factored into the formula.

Sizzle Selectors

Sizzle is a JavaScript selector library that offers powerful ways to select elements. You can select based off text contained or not contained within elements, the existence of child elements inside a parent, or if those elements do not exist.

> Sizzle selectors can only be used in the **Element Selector** field and are not supported in CSS.

For more information about Sizzle, including the full list of selectors supported, refer to jQuery's Wiki on GitHub.

:has

The `:has()` selector matches when a parent element contains at least one element that matches the specified selector. For example, `div.parent:has(div.child)` selects `div.parent` only if a `div.child` element exists anywhere among its descendants, not just as a direct child.

:contains

The `:contains()` selector matches when a parent element has text that matches the string. For example `div.parent:contains(string)` selects `div.parent` only if it has text directly in it or anywhere among its descendants that matches the string. The string is case-sensitive and needs to match only part of the text content.
A `div:contains(order)` selector selects `<div>` elements containing the text string `order`. It also matches `<div>` elements that contain the word *border* because the string still matches part of the word.

:not

The `:not()` selector matches all elements that do not match the given selector.

You can specify that the matching selector is not a specific type. For example, `.box:not(div, p)` selects all elements with `box` class except `<div>` and `<p>` elements. You can also format the selector with spaces rather than comma separation:

```
.box:not(div p)
```

You can specify other attributes, IDs, or classes, such as `div:not(.box)`, which selects `<div>` elements except ones with `box` class.

:not in CSS

You can use the `:not()` selector in CSS as well. In the example below, all elements with the `box` class get a red background *unless* they are `<p>` tags.

```
.box:not(p) {
  background: red;
}
```

:eq

The `:eq(n)` selector selects an element with an index number equal to *n*. Unlike pseudo-class selectors, the positional index for Sizzle begins at 0 (zero), so `.main > div:eq(2)` matches the third first-level `<div>` within `.main` rather than the second.

:nth

Similar to `eq`, the `:nth(n)` selector matches an element with a number *n*.

:odd and :even

Because the positional index begins at 0, `:odd` and `:even` work counterintuitively.

li:odd

The `li:odd` selector matches the second, fourth, sixth, and other even-numbered list items.

li:even

The `li:even` selector matches the first, third, fifth, and other odd-numbered list items.

:gt

This selector selects all elements *greater than* the specified number. Remember, the positional index is 0. For example, the `.box:gt(1)` selector would select all `.box` occurrences after the second instance.

:lt

This selects all elements **less than** the specified number. Remember, the positional index is 0.
For example, the `.box:lt(4)` selector would select the first four instances of `.box`. Remember, 0 through 3 are the first four instances.

:first

The `:first` pseudo-class is equivalent to `:eq(0)`. You could also write it as `:lt(1)`.

> The difference between `:first` and `:first-child` is that `:first` *only* matches a single element. `:first-child` can match more than one element, including one for each parent.

In the example `div p:first`, the selector matches *only* the first `<p>` element within a `<div>`.

:last

The `:last` selector selects a single element by filtering *the current sizzle or jQuery collection* and matching the last element within it.

> The current collection can change based on interactive elements within a page.

In the example `div p:last`, the selector matches *only* the last `<p>` element within a `<div>` based on the current Sizzle or jQuery collection.

> When you use selectors in a global CSS stylesheet, the styling applies to all elements that they match. When used in Monetate actions in the **Element Selector** field, they only match the first instance unless you enable **Select Multiple Elements**.

For a complete list of CSS selectors, refer to W3 Schools. For an example of various CSS selectors, refer to the Selector Tester application by W3 Schools.

> Some CSS selectors are not supported by earlier versions of Internet Explorer. For a more in-depth list, refer to the compatibility chart available on Quirksmode.