**Utility**

|  |  |
| --- | --- |
| u-utilityName Syntax: u-<utilityName>  Notation: camel case | |
| Definition: | *Low-level structural, positional, and visual traits. Utilities can be applied directly to any element within a component. They modify one presentational aspect and are broadly applicable. They are different from a modifier because they aren’t specific to one particular component. Utilities always have a consistent meaning in whichever context they are used.* |
| *Example 1: How various utilities can be used to create a simple layout.* | |

<div class=**"u-cf"**><!—clearfix (contain floats) -->

<a class=**"u-objLeft"** href=**"{url}"**><!-- float left with right margin -->

<img class=**"u-block"** src=**"{src}"** alt=**""**><!-- display block -->

</a>

<p class=**"u-sizeFill u-textBreak"**><!-- fill remaining space; break long strings -->

…

</p>

</div>

|  |
| --- |
| *Example 2: Very occasionally, a utility will also need to apply styles to a descendant element. The descendant is targeted using a class of the form: u-utilityName-descendantName:* |

<a class="u-linkComplex" href="{url}">

<span class="u-linkComplex-target">Underline on hover.</span>

No underline on hover.

</a>

**Component namespace**

|  |  |
| --- | --- |
| **namespace (optional)**  Syntax: [<namespace>-]<ComponentName>[--modifierName|-descendantName]  Notation: lowercase | |
| Definition: | *If necessary, components can be prefixed with a namespace. For example, you may wish to avoid the potential for collisions between libraries and your custom components by prefixing all your components with a namespace. This makes it clear, when reading the HTML, which components are part of your library.* |
| *Example 1* | |

.twt-Button { /\* … \*/ }

.twt-Tabs { /\* … \*/ }

**Component**

|  |  |
| --- | --- |
| **ComponentName**  Syntax: [<namespace>-]<ComponentName>[--modifierName|-descendantName]  Notation: pascal case (nothing else in the HTML/CSS uses pascal case) | |
| Definition: | *Components are UI patterns. They usually need to style multiple elements within their HTML tree. Components have several benefits when writing CSS and reading HTML:*   * *It keeps the specificity of selectors low.* * *It helps to decouple presentation semantics from document semantics.* * *It helps to distinguish between base components, modifiers of components, and child elements.* |
| *Example 1* | |

<article class=**"MyComponent"**>

…

</article>

**Component modifier**

|  |  |
| --- | --- |
| **ComponentName--modifierName**  Syntax: [<namespace>-]<ComponentName>[--modifierName|-descendantName]  Notation: camel case (separated from the component name by two hyphens) | |
| Definition: | *A component modifier is a class that modifies or extends the presentation of the base component in some form. The class should be included in the HTML in addition to the base component class (a multiple class system). It can be seen as a specific incarnation of a component (as in for example: Human Human--female).* |
| *Example 1* | |

/\* Core button \*/

.Button { /\* … \*/ }

/\* Default button theme \*/

.Button--default { /\* … \*/ }

|  |
| --- |
| *Example 2* |

<button class="Button Button--default" type="button">…</button>

**Component descendant**

|  |  |
| --- | --- |
| **ComponentName-descendantName**  Syntax: [<namespace>-]<ComponentName>[--modifierName|-descendantName]  Notation: camel case | |
| Definition: | *A component descendant is a class that is attached to a descendant node of a component. It's responsible for applying presentation directly to the descendant on behalf of a particular component. It “belongs” to the component, and does not have meaning outside of its context (as in: Human Human-hand Human-foot).* |
| *Example 1* | |

<article class=**"Tweet"**>

<header class="Tweet-header">

<img class="Tweet-avatar" src="{{src}}" alt="{{alt}}">

…

</header>

<div class="Tweet-body">

…

</div>

</article>

**Component state**

|  |  |
| --- | --- |
| **ComponentName.is-stateOfComponent**  Syntax: is-stateOfComponent  Notation: camel case | |
| *Definition:* | *Use the component state for state-based modifications of components. JS can add/remove these classes. This means that the same state names can be used in multiple contexts, but every component must define its own styles for the state by being scoped to the component.*  ***Never style these classes directly; they should always be used as an adjoining class.*** |
| *Example 1* | |

.Tweet { /\* … \*/ }

.Tweet.is-expanded { /\* … \*/ }

|  |
| --- |
| *Example 2* |

<article class="Tweet is-expanded">

…

</article>

**Scoping to specific breakpoints**

|  |  |
| --- | --- |
| **[n]-utilityName or [n]-ComponentName**  Syntax: [size]-[Component or u-utility]  Notation: camel case | |
| *Definition:* | *To scope utility or component styles to a Media Query breakpoint, use a* .size- *class name prefix to denote which variant the utility is scoped to. A size naming scheme could be clothing sizes;* [xxs, xs, s, m, l, xl, xxl]*.* |
| *Example 1* | |

<div class="xs-u-alignLeft xxl-u-textCenter">...</div>

**Javascript specific classes**

|  |  |
| --- | --- |
| **js-someName**  Syntax: .size-[scopedComponent or utility]  Notation: camel case | |
| *Definition:* | Use of the id attribute and js-\* class names are reserved for JavaScript-only use. Application-specific data or content can be stored in data-\* attributes.  **Other CSS must not use js-\* classes in selectors.** |
| *Example 1: A dedicated JavaScript utility class to which behaviour is bound. It is independent of any specific UI component.* | |

<a class=**"js-showProfile"** data-username=**"necolas"** href=**"{url}"**>**...**</a>

Zie ook: <https://github.com/suitcss/suit> en <https://github.com/coreybruyere/530-framework/blob/master/README.md> (nog verwerken in documentatie)