Appendix B

LinterDSL documentation.

1. Linter (root element)

The Linter object is the main element of any LinterDSL document. It holds the collections of checks and reusable selector definitions.

Syntax :

```
linter <no name> for <no supportedFramework> version <no supportedVersion>

reusable selector definitions:

  << ... >>

validation checks:

  << ... >>
```

Parameters:

- name (string): name value of the linter object. The generated file will be named the same as the linter object.
- supportedFramerwork (string): place to specify the framework name on which the linter will be used.
- supportedVersion (string): version of the supported framework. Both parameters (supportedFramework and supportedVersion) will be placed in a comment of the generated linter for informational purposes.

Collections:

- selectorDefinitions (Selector[0..n]): holds Selector objects that can be used throughout the documents.
- checks (Check[0..n]): collection of validation rules for the framework.

2. Check

Check is an abstract concept of the language which holds the common elements of its subtypes.

Syntax part common to all subtypes:

```
<alias of the subtype> check with id <no id> raises <no type>
  elements << ... >>
  <place for additional parameters and collections depending on the subtype>
  custom message:  <no customMessage>
```

Parameters common to all subtypes:

- id (integer): unique identifier of the check.
- type (CheckType): enumeration which can be set to one of the following: ‘Error’, ‘Warning’.
- customMessage (string): custom message can override the automatically generated message upon violation. To use the automatic message set the value to the ‘-‘ sign.
- applyTo (Selector[1..n]): describes the set of elements to which the check will be applied.

2.1 AllowedChildrenCheck

Ensures that all children of elements matching one of the selectors specified in the applyTo collection match one of the selectors specified in the allowedChildren collection.

Alias: allowed children check

Syntax:

```
allowed children check with id <no id> raises Error
  elements <applyTo>
  can only have children <allowedChildren>
  custom message: <no customMessage>
```

Parameters:

- allowedChildren (Selector[1..n]): specifies the set of allowed children for the investigated element.

Default violation message:

- InappropriateChildren{type} (check id: {id}): elements which satisfy {applyTo} can only have children matching {allowedChildren}. Elements with inappropriate children: [list of invalid elements]

2.2 DirectParentCheck

Ensures that an element is placed within a right parent.

Alias: direct parent check

Syntax:

```
direct parent check with id <no id> raises <no type>
  elements <applyTo>
  must have a parent <allowedParents>
  custom message: <no customMessage>
```

Parameters:

- allowedParents (Selector[1..n]): specifies the set of allowed direct parents for the investigated element.

Default violation message:

- DirectParent{type} (check id: {id}): elements which satisfy {applyTo} can only be direct children of elements matching: {allowedParents}. Elements with inappropriate parents: [list of invalid elements]

2.3 InvalidElementCheck

Raises a Warning/Error if it finds elements matching any selectors given in the {applyTo} set.

Alias: invalid element check

Syntax:

```
invalid element check with id <no id> raises <no type>
  elements <applyTo>
  are not valid in the framework
  custom message: <no customMessage>
```

Default violation message:

- InvalidElement{type} (check id: {id}): elements matching {applyTo} are not valid in {Linter.supportedFramework}. InvalidElements: [list of invalid elements]

2.4 MissingTagCheck

Raises a Warning/Error if no elements matching any selectors specified by the {applyTo} set are found within the document.

Alias: missing tag check

Syntax:

```
missing tag check with id <no id> raises <no type>
  framework requires an element <apoplyTo>
  custom message: <no customMessage>
```

Default violation message:

- MissingElement{type} (check id: {id}): presence of an element matching {applyTo} is required by {Linter.supportedFramework}.

2.5 MisuseCheck

Applies additional requirements for the elements picked up by the {applyTo} selectors by listing another set of selectors that must be satisfied by them.

Alias: element misuse check

Syntax:

```
element misuse check with id <no id> raises <no type>
  elements <applyTo>
  must also be elements <required>
  custom message:  <no customMessage>
```

Parameters:

- required (Selector[1..n]): specifies a set of additional requirements for the selected elements.

Default violation message:

- ElementMissuse{type} (check id: {id}): elements which satisfy {applyTo} must also match {required}. Misused elements: [list of invalid elements]

2.6 PredecessorsCheck

Ensures that all elements matching one of the {applyTo} selectors are placed as a descendant of an element specified by another set of selectors.

Alias: predecessors check

Syntax:

```
predecessors check with id <no id> raises <no type>
  elements <applyTo>
  must have an ancestor <requiredPredecessor>
  <no generations> generations up
  custom message: <no customMessage>
```

Parameters:

- requiredPredecessor (Selector[1..n]): specifies the set of selectors that must be matched the required ancestor.
- generations (integer): specifies the how far apart in terms of generations the elements should be to the required ancestors. If the value is set to ‘0’ all previous generations will apply.

Default violation message:

- Predecessor{type} (check id: {id}): elements which satisfy {applyTo} must be descendants of an element matching: {requiredPredecessor}. Misplaced elements: [list of invalid elements]

2.7 RequiredChildCheck

Ensures that all elements matching one of the {applyTo} selectors are have a required child element specified by another collection of selectors.

Alias: required child check

Syntax:

```
required child check with id <no id> raises <no type>
  elements <applyTo>
  must have <no uniqueFlag> one child <requiredChild>
  at position <no position>
  custom message: <no customMessage>
```

Parameters:

- requiredChild (Selector[1..n]): specifies the set of selectors that must be matched by the required child element.
- uniqueFlag (UniqueFlag): enumeration mapping strings ‘exactly’ -> true and ‘at least’ -> false. If set to ‘exactly’, the rule will be wiolated if more than one elements match the {requiredChild} selectors.
- Position (integer): specifies the position of the child. If set to ‘0’ the can appear at any index of the investigated element. If set to ’-1’ the specified child must be also the last one.

Default violation message:

- RequiredChildrenError (check id: 46): elements which satisfy .carousel must have exactly one child matching {applyTo}. Elements missing required children: [list of invalid elements]

2.8 Siblings check

This check allows for validating the siblings of the selected elements in various ways.

Alias: siblings check

Syntax:

```
siblings check with id <no id> raises <no type>
  elements <applyTo>
  can <no condition> <allowedSiblings>
  custom message: <no customMessage>
```

Parameters:

- allowedSiblings (Selector[0..n]): applies requirements for the siblings under consideration
- condition (SiblingsCheckType): an enumeration allowing for the following string values:
  -‘not have next siblings’
  -‘not have previous siblings’
  -‘only appear after elements’
  -‘only appear before elements’
  -‘only have next immediate sibling’
  -‘only have previous immediate siblings’
  -‘only appear with at least one sibling’

During the code generation process these options are mapped to appropriate actions. If one of the ‘not have’ options are the allowedSiblings collection must be empty, otherwise it must contain at least one selector. The generated message is different for every condition.

3. SelectorDefinition

SelectorDefinition objects can be children of the root element of LinterDSL. They can hold a collection of Selector objects which can then passed by reference to a Check or another definition with use of a ReferenceSelector. All names of selector definitions must be unique.

Alias: selector definition

Syntax:

```
selector <no name> filters elements <selectors>
```

4. Selector

Selector objects represent CSS selectors without the need of the CSS syntax, using a more descriptive one instead. Selector is an abstract concepts with many subtypes that can be used in the model. Every collection of Selector objects mentioned so far uses ‘or’ as a separator, the only exception will be the GroupSelector.

4.1 AttributeSelector

Translates to standard CSS attribute selector ([attributeName=value]).

Alias: attribute

Syntax:

```
with attribute <no attributeName> = <no value>
```

Parameters:

- attributeName (string): name of the HTML attribute.
- value (string): value of the HTML attribute. If set to '*' the value will not be specified. To look for value = '*' use '/*'.

4.2 ClassSelector

Translates to standard CSS class selector (.className).

Alias: class

Syntax:

```
with class <no value>
```

Parameters:

- value (string): the HTML class.

4.3 TypeSelector

Translates to standard CSS tag selector (tagName).

Alias: type

Syntax:

```
of type <no value>
```

Parameters:

- value (string): the HTML tag.

4.4 NotSelector

Translates to standard CSS not selector (:not(selector)).

Alias: not

Syntax:

```
not <selector>
```

Parameters:

- selector (Selector): other selector to be inverted.

4.5 GroupSelector

Concatinates the generated CSS selectors.

Alias: group

Syntax:

```
not <selector>
```

Parameters:

- selectors (Selector[1..n]): selectors to be concatinated.

4.6 CustomSelector

Allows to input a custom CSS selector with the standard CSS syntax.

Alias: custom

Syntax:

```
matching <no string>
```

Parameters:

- string (string): the CSS selector.

4.7 ClassPatternSelector

Allows to declare a selector filtering elements which match a regular expression.

Alias: pattern

Syntax:

```
with class matching pattern <no regularExpression>
```

Parameters:

- regularExpression (string): elements must have a class which matches this parameter to satisfy this selector.

4.8 ReferenceSelector

Body of a reference selector will be replaced with the body of the referenced selector, allowing for reusing the same definitions.

Alias: reference

Syntax:

```
matching <no selector>
```

Parameters:

- selector (reference to SelectorDefinition): selector definition passed by putting its name.