Tricss

You're keen on adding your own properties to Css?
Or support for a -webkit-abc property in Firefox?
Your imagination is the boundary!

Project Page: <http://chrisbk.de/repository/tricss/>
Git Examples: <http://chrisbk.de/repository/tricss/examples/>

Note: This is only a short summary. For further details visist the soon-to-be-released project page.

Properties

Tricss.Properties is a Hash containing all CSS properties used in Tricss. Furthermore this object is used to add “this property changes” events.

Getter

If a declaration is returned through *Tricss.Rule.getDeclaration*, this function is invoked to modify the result.

Setter

When a declaration is set by calling *Tricss.Rule.setDeclaration*, the script uses this function to alter the input.

Initial Value

If a style is not yet set via a *Tricss.Rule* (or *Element.setStyle*), this value is returned.

Default

Referenced to as *default* is an object containing an empty getter & setter function (*$empty*) and an empty initial value (*‘’*).

Tricss.Properties Function: get

Gets a Tricss CSS property. If no property was set, Default is returned.

Syntax

```
Tricss.Properties.get(property);
```

Arguments

1. property (*string*) - The CSS property.

Returns

(*object*)
- getter (*function*) - Getter
- initial (*mixed*) - Initial
- setter (*function*) - Setter

Examples

```
alert('Initial value is ' + Tricss.Properties.get('my-secret-property').initial);
```

Tricss.Properties Function: set

Sets a Tricss CSS property. The object is extended with Default.

Syntax

Tricss.Properties.get(property, obj);

Arguments

1. property (*string*) - CSS property.
2. options (*object*)

Options

(*object*)
    getter (*function*) - Getter
    initial (*mixed*) - Initial
    setter (*function*) - Setter

Returns

(*object*) Tricss.Properties

Examples

```
Tricss.Properties.set('my-secret-property', {
    initial: '20px'
});
```

Tricss.Properties Function: observe

Adds an observer which is triggered when the specified property changes.

Syntax

Tricss.Properties.observe(property, fn);

Arguments

1. property (*string*) - CSS property to observe.
2. fn (*function*) - Callback function.

Callback Arguments

1. element (*element*) - The element affected.
2. value (*mixed*) - The new value.

Returns

(*object*) Tricss.Properties

Examples

```
Tricss.Properties.observe(‘maginify’, function(element, value){
    element.setStyle(‘font-size’, element.getStyle(‘font-size’).toInt() * value);
});
```

Tricss.Properties Function: unobserve

Removes an observer described above.

Syntax

Tricss.Properties.unobserve(property, fn);

Arguments

1. property (*string*) - CSS property.
2. fn (*function*) - Callback function to remove.

Returns

(*object*) Tricss.Properties

Examples

Tricss.Properties.unobserve(‘maginify’, myFn);

Document

This script loads all CSS style sheets included through *<style>* or *<link>* elements. This markup is parsed, converted into rules and applied to the HTML document like normal *Tricss.Rules*.

Ready

When everything described above is loaded, Tricss or *Tricss.Document* is *ready*.

Notes

Css.Properties observers are fired first when Tricss is ready.

tricss:ready Event

This event is similiar to the *domready* event. It's fired when Tricss is ready.

Examples

```
document.addEvent('tricss:ready', function(){
   alert('Everything parsed and added. Tricss is ready now :)');
});
```

Tricss.Document Variable: ready

Wheather the *Tricss.Document* is ready or not.

Returns

(*boolean*) Ready?

Examples

```
alert('Tricss is ' + (Tricss.Document.ready) ? '' : 'not' + ' ready');
```

Tricss.Document Variable: rawRules

Contains all rules added with *Tricss.Document.addCss*.

Returns

(*array*) The raw rules.

Examples

```
Tricss.Document.rawRules.each(function(rule){
    alert('Selector: ' + rule.selector);
    alert('Body: ' + rule.body);
});
```

Tricss.Document Variable: rules

Despite *Tricss.Document.rawRules* this property contains only rules with declarations set in *Tricss.Properties*. Only rules with at least one such declaration are included.

Returns

(*array*) The rules.

Examples

```
alert('The document has ' + Tricss.Document.rules.length + ' rules.');
```

Tricss.Document Function: addCss

Adds CSS markup. The markup is parsed, converted into rules and applied.

Syntax

Tricss.Document.addCss(css);

Arguments

1. css (*string*) - The events ..

Returns

(*object*) *Tricss.Document*.

Examples

Tricss.Document.addCss("div#container {text-align: center;}");

Tricss.Document Function: addStylesheet

Adds either a CSS *<style>* or *<link>* element. Calls *Tricss.Document.addCss* with the markup inside the element or linked stylesheet.

Syntax

Tricss.Document.addStylesheet(element, fn);

Arguments

1. element (*element*) - The element
2. fn (*function*) - The callback executed after the CSS was added.

Returns

(*object*) *Tricss.Document*.

Examples

```
Tricss.Document.addStylesheet($(link#globalStyles), function(){
    alert('added');
});
```

See Also

>> Tricss.Document.addCss

Event

Element Method: addTricssEvent

This function adds an event listener to an element.
Natively supported events are the dynamic css pseudos *active*, *focus* and *hover*.

Syntax

```
myElement.addTricssEvent(events, when, fn);
```

Arguments

1. events (*array or string*) - The events ..
2. when (*string, optional*) - When the event should trigger. Either when the event occures (‘enter’) or when left (‘leave’). Defaults to ‘enter’.
3. fn (*function*) - The function which should be executed.

Returns

(*element*) This Element.

Examples

```
$(‘myElement’).addTricssEvent([‘hover’, ‘focus’], function(){
	alert(‘hovered and focused’);
});

$(‘myElement’).addTricssEvent(‘hover’, ‘leave’, function(){
	alert(‘unhovered’);
});
```

See Also

W3C / MDC Dynamic Pseudos
http://docs.mootools.net/Element/Element.Event#Element:addEvent

Element Method: removeTricssEvent

Similiar to Element.addTricssEvent, but instead of adding, this method removes the event listener.

Syntax

```
myElement.removeTricssEvent(events, when, fn);
```

Arguments

1. events (*array or string*) - The name of the events.
2. when (*string, optional*) - When the event should trigger. Either when the event occures (‘enter’) or when left (‘leave’). Defaults to ‘enter’.
3. fn (*function*) - The function which should be removed.

Returns

(*element*) This Element.

Examples

```
$(‘myElement’).addTricssEvent([‘hover’, ‘focus’], function(){
	alert(‘hovered and focused’);
});

$(‘myElement’).addTricssEvent(‘hover’, ‘leave’, function(){
	alert(‘unhovered’);
});
```

See Also

Element.addTricssEvent
http://docs.mootools.net/Element/Element.Event#Element:removeEvent

Own Events

You can also add your own events. Just extend the *Tricss.Events* hash.

```
Tricss.Events.set({‘myEvent’, {
	enter: 'enterMyEvent',
	leave: 'leaveMyEvent'
});
```

Example

```
Tricss.Events.set({‘active’, {
	enter: 'mousedown',
	leave: 'mouseup'
});
```

Parser

Tricss.Parser Function: declarations

Parses CSS declarations into an object. Declarations are usually inside the CSS rule body.

Syntax

```
Tricss.Parser.declarations(css);
```

Arguments

1. css (*string*) - The markup to parse.

Returns

(*object*) - the declarations.
        (*object*)
                (*boolean*) important - is the declaration *!important* ?
                (*string*) value - the declaration’s value.

Examples

```
Tricss.Parser.declarations(“font-size: 14px; -custom-property: customValue !important;”);

/* returns:
{
   ‘font-size’: {
      important: false,
      value: ‘14px’
   },
   ‘-custom-property’: {
      important: true,
      value: ‘customValue’
   }
}
*/
```

Tricss.Parser Function: rules

Parses CSS rules into an object.

Syntax

Tricss.Parser.rules(css, parseDeclarations);

Arguments

1. css (*string*) - The markup to parse.
2. parseDeclarations (*boolean, optional*) - should the declarations be parsed?, defaults to false.

Returns

```
(array) - contains the rules.
   (object)
      (string) body - the rule’s body.
      (object or boolean) declarations - if parseDeclarations is true, this property contains
                                              the declaration object, else is set to false.
      (string) selector - the rule’s selector.
```

Examples

```
Tricss.Parser.parse(“div#a:hover { color: orange; }”, true);

/* returns:
[
   {
      body: ‘ color: orange; ‘
      declarations: {
         color: {
            important: false,
            value: ‘orange’
         }
      },
      selector: ‘div#a:hover’
   }
]
*/
```

Notes

If multiple selectors (*selectorA, selectorB ...*) are used, a rule for each selector is returned. Note that this is against standards, but makes useing way easier.

Rule

Class: Tricss.Rule

Represents a CSS rule.

Syntax

new Tricss.Rule(argA, declarations);

Arguments

1. argA (*element, string or object*) - See ›› Construction below.
2. declarations (*object, optional*) - The rule's declarations.

Construction

This class can either be constructed with a selector or with an element. The selector can either be a string or a *Tricss.Selector*.

Returns

(*object*) - the declarations.
    (*object*)
        (*boolean*) important - is the declaration *!important* ?
        (*string*) value - the declaration's value.

Examples

```
Tricss.Parser.declarations("font-size: 14px; -custom-property: customValue !important;");

/* returns:
{
   'font-size': {
      important: false,
      value: '14px'
   },
   '-custom-property': {
      important: true,
      value: 'customValue'
   }
}
*/
```

See Also

>> Properties
>> Properties#observing
>> Selector

Importance

Throughout Tricss the following importances are used:

1 - “normal” declaration.
2 - the declaration is declared as !important.
3 - inline style.

Tricss.Rule Method: getDeclaration

Gets a declaration - value and importance - of a specific property.

Syntax

```
myRule.getDeclaration(property);
```

Arguments

1. property (*string*) - The declaration’s property.

Returns

(*object*) - The declarations (see below).

Each property - representing the declaration’s property - has the following value:

(*object*)
   (*number*) importance - The declaration’s importance.
   (*mixed*) value - The declaration’s value.

Examples

```
myRule.getDeclaration(‘font-size’);

/* may return:
{
   importance: 2,
   value: ‘13px’
}
*/
```

Tricss.Rule Method: setDeclaration

Sets a declaration.

Syntax

```
myRule.setDeclaration(property, value, importance);
```

Arguments

1. property (*string*) - The declaration’s property.
2. value (*mixed*) - The declaration’s value.
3. importance (*number, optional*) - The declaration’s importance, defaults to 1.

Returns

(*object*) The rule.

Examples

```
myRule.setDeclaration(‘font-family’, ‘Verdana, Arial’, 3);
myRule.setDeclaration(‘font-weight’, 800);
```

Tricss.Rule Method: setDeclarations

Sets declarations.

Syntax

```
myRule.setDeclarations(declarations);
```

Arguments

1. declarations (*object*) - The declarations (see below).

Each property - representing the declaration’s property - has the following value:

```
{
    importance (number, optional) - The declaration’s importance.
    value (mixed) - The declaration’s value. Defaults to 1.
}
```

or

```
[
    value (mixed) - The declaration’s value.
    importance (number, optional)  - The declaration’s importance. Defaults to 1.
]
```

or

value (*mixed*) - The declaration’s value. Importance is 1.

Returns

(*object*) The rule.

Examples

```
myRule.setDeclarations({
    ‘position’: {
        value: ‘relative’
    },
    top: {
        importance: 3,
        value: ‘relative’
    }
    ‘margin-left’: [‘0px’, 2],
    ‘padding’: ‘2px 2px 4px 0px’,
});
```

See Also

>> Tricss.Rule.setDeclaration

Tricss.Rule Method: getElements

Returns the elements the rule is matching to.

Syntax

```
myRule.getElements();
```

Returns

(*array*) The elements the rule is matching to.

Examples

```
myRule.getElements().each(function(element){
    alert(element);
});
```

See Also

>> Tricss.Selector.getElements

Tricss.Rule Method: getSpecificity

If the rule was initialized with a selector, the selector’s sepcificity is returned. When an element was passed, the specificity *0* is returned.

Syntax

```
myRule.getSpecficitiy();
```

Returns

(*number*) The rule’s specificity.

Examples

```
// myRuleB must be declared after myRuleA!
var overrides = (myRuleA.getSpecificity() <= myRuleB.getSpecificity());
alert(‘myRuleB overrides myRuleA? ‘ + overrides);
```

See Also

>> Tricss.Selector.getSpecificity.
http://www.w3.org/TR/REC-CSS2/cascade.html#specificity

Selector

Class: Tricss.Selector

Represents a CSS Selector.

Syntax

```
new Tricss.Selector(selector);
```

Arguments

1. selector (*string*) - The selctor.

Examples

```
new Tricss.Selector(‘a.link div’);
```

Tricss.Selector Method: addEvent

This class delegates to Tricss.Rule.Element or Tricss.Rule.Selector.

Syntax

```
new Tricss.Selector(event, fn);
```

Arguments

1. event (*string*) - Either *complies* or *uncomplies*.
2. fn (*function*) - The callback function.

Event

The event *complies* is fired when the selector expression is true. E.g. when the selector contains *#myElement:hover* and the mouse enters *#myElement*.

Contrary the object fires *uncomplies*, when the expression is no longer true. Here when the mouse leaves.

Examples

```
var mySelector = new Selector(‘a:hover’);

mySelector.addEvent(‘complies’, function(element){
    alert(‘You entered the link: ‘ + element);
});

mySelector.addEvent(‘uncomplies’, function(element){
    alert(‘You left the link: ‘ + element);
});
```

Notes

This class extends *Events*. You can use all methods described in >>Events.

Tricss.Selector Method: getElements

Returns the elements matching the selector.

Syntax

```
mySelector.getElements();
```

Returns

(*array*) The matching elements.

Examples

```
mySelector.getElements().each(function(element){
    element.setStyle(‘padding-left’, ‘20px’);
});
```

See Also

>> Tricss.Rule.getElements

Tricss.Selector Method: getSpecificity

Returns the selector’s sepcificity.

Syntax

```
mySelector.getSpecficitiy();
```

Returns

(*number*) The specificity.

Examples

```
var mySelector = new Selector(‘div.asdf p#ghjk’);
alert(mySelector. getSpecficitiy() == 112); // alerts true
```

See Also

>> Tricss.Rule.getSpecificity.
http://www.w3.org/TR/REC-CSS2/cascade.html#specificity

Styles

Extends *Element* in order to use Tricss' rules with *Element.getStyle* and *Element.setStyle*.