Soda Selectors

How to select elements from the Soda DOM

What's a selector?

A *Soda selector* is a string used return a set of elements to test against. They function much like CSS selectors and are used to find elements at a granular level.

Soda uses selectors to match elements based on their *properties*. Given a selector, Soda will always return a set (zero to *n*) elements that "match" the selector based on the available DOM Tree (e.g. page, window, or screen).

There are four basic property keys used to find elements: *id*, *name*, *label*, and *value*. Using a single character, we've created a mapping from element property to character (which we call an *operator*):

The 4 Basic Selector Operators

Element Property	**Id**	**Name**	**Label**	**Value**
Character	# (Pound)	^ (Caret)	. (Period)	@ (At)

Selector Examples

Using the four basic operators: id, name, label and value…

```
// Select all elements that have the property id equal to "textfield:0"
#textfield:0

// Select all elements that have the property name equal to "userName"
.userName

// Select all elements that have the property label equal to "userName"
^username

// Select all elements that have the property value equal to "Username"
@Username

// *** If a property has special characters or spaces, wrap it in curly braces ***
// Select all elements that have the property value equivalent to "Forgot password?"
@{Forgot password?}
```

Compound Selectors

Using the four basic operators, we can combine them and use “compound selectors”

```
// Select all elements that have the property id equal to “textfield:0" and a name property
// equal to “userName”
#textfield:0.userName

// Select all elements that have the property id equal to “textfield:0", a name property
// equal to “userName” and a value equal to “Username”
#textfield:0.userName@Username

// Or using curly braces
#{textfield:0}.{userName}@{Username}

// Select all elements that have a name property which is an array, with multiple names
// Suppose the element has the name values [“foo”, “bar”, “baz”]
.foo
.foo.bar
.foo.bar.baz
.baz.bar.foo
.baz.foo
...
```

Selector Hierarchy

Spaces in selectors represent hierarchy, where parentage follows from left to right.

You can use selector hierarchy to match elements which have identifiable parents.

```
// Select all elements that have the property id equivalent to “textfield:0" and the parent
// with an id of “window:0”
#window:0 #textfield:0

// Select all elements that have the property name equivalent to “userName" and the parent
// with an id of “window:0”
#window:0 .userName

// Select all elements that have the property label equivalent to “userName" and the parent
// with a label of “elementLabel”
^elementLabel ^username

// More examples...
#window:0 #textfield:1 @Username
#window:0 #textfield:1 ^.userName
#id #id ^label .name @value
```

The Wildcard Operator

Use the wildcard operator (*) to select all descendants.

The asterisk (*) is the wildcard selector operator and selects all elements at or below the specified “level.”

```
// Select all elements (level 0, “root”)
*

// Selects all elements that are a child of root elements (level 1)
* *

// Selects all elements all elements that are grand children of the root level (level 2)
* * *

// etc. etc.
* * * *
* * * * *

// You can use the wildcard with other selectors
// Selects all elements that are children of elements with their property name equivalent to
// “userName”

.userName *

// More examples
#window:0 * .userName
#window:0 * *
* * @Username
@elementWithThisValue ^{elementWithThisLabel which has $pecial c4aracters}  *
```

The Direct Descendant Operator

Use the direct descendant operator (>) to select immediate children.

The direct descendant operator differs from the wildcard operator as the wildcard selects all children, grandchildren, and nth-grand children of the given set. The direct descendant operator only returns the direct children of each of the elements in the given set

```
// Select elements that are children of the root
>

// Select the immediate children of an element with the value “Username”
@Username >

// Select the children of the children of an element with the id “window:0”
#window:0 > >

// Select all elements that have children
// The star returns all elements, then the > selects the children of all elements.
// So if an element has children, it is returned
* >

// Select all non-root elements
> *
// Or alternatively...
* *
```

The Direct Ascendant Operator

Use the direct Ascendant operator (<) to select immediate parents.

The direct ascendant operator grabs the parents of the elements in the given set.

```
// Select the parents of root-level elements
// This will always return an empty set
<

// Select the parents of all elements which have the property name equal to ".userName"
.userName <

// Select the grandparents all elements which have the property name equal to ".userName"
.userName < <

// Select the grandparents all elements which have the property name equal to ".userName"
// > < is a basically a no-op
.userName < < > <

// Selects all elements with the value "Username", from the children of the elements with the
// property label equal to "someElement"
^someElement > @Username
```

Property Filters

Use property filters to select elements based on *any* element property

Property filters are appended to selectors using brackets, in the format *[property(operator)value]*. There are ten property filter operators

The 10 Property Filter Operators

Operator	Description
Equal To	=
Not equal to	!=
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Like (regular expression)	~
Not like (negated regular expression)	!~
Has*	?=
Does not have*	?!

** Only operates on objects (arrays and pure objects)—operations on strings will return false.*

Property Filter Examples

Using the 8 property filter operators

```
// Select all elements with the property name equal to “username” and filter them by value
// Note: string values should be quoted
.userName[value=’some value’]
.userName[value!=’some value’]
.userName[value>0]
.userName[value<0]

// Select all elements which have the parent with the label parent, filtered by window position
^parent *[rect.origin.x=0]
^parent *[rect.origin.y=4.17]

// Select all elements with the name userName which have the parent with the label parent,
// filtered by size
^parent .userName[rect.size.width=100]
^parent .userName[rect.size.height=1345.12]

// Select all enabled/disabled elements
// Note, Boolean values should *not* be quoted
*[enabled=true]
*[enabled=false]

// Select all visible/hidden
*[visible=true]
*[visible=false]

// Select all elements of type ‘textfield’
*[type=’textfield’]

// Select all enabled and visible elements
*[enabled=true][visible=true]

// Select all elements whose values matches the regular expression /admin.*/
*[value~’admin.*’]

// Select all elements whose value is not “some value”
*[value!=’value’]

// Select the children of the element with the id “window:0” whose x position is greate than 500
#window:0 >[rect.origin.x>500]

// Select an element who has multiple names (classes)
// This is only valid for web
.nameOne[name?=’name2’][name?=’name3’]

// Select all elements who don’t have the name (class) “noName”
*[name?!=’noName’]

// Note ?= and ?! operate on sub-objects of the element
// So, if desired, you could do...
// Grabs all elements with a parent that has a label
*[parent?=’label’]

// Grabs all elements with a parent that does not have a name
*[parent?!’name’]
```

```
// Combine property filters!
.userName[value='some value'][label='some label]
@Username[enabled=true][visible=true][value!='some value']
```

The Nth Property Filter

The "nth" property filter is a special filter that chooses the *nth* element from the returned set.
When using the *nth* property filter, Soda will first select all elements which match the given selector, then return the *nth* element in the set (if it exists). Remember Soda always returns a set, so using *nth*, you'll either get a set of size one (if the *nth* element exists) or zero (if it doesn't).

```
// Select the first element in the returned set of elements with the name "userName"
// Note: nth is zero-based
.userName[nth=0]

// Use "first" and "last" to select the first and last elements, respectively
.userName[nth='first'] // Returns the first element with the name "userName"
.userName[nth='last']  // Returns the last element with the name "userName"

// Go crazy!
// The following ridiculous selector just selects #window:0
#window:0[type='window'] > < *[nth=0] < >[type!='foo'] *[nth=1] <[type~'.*'] <[type~'window']
```