programcreek.com

Java Code Example org.w3c.css.sac.Selector

22-28 minutes

The following are top voted examples for showing how to use org.w3c.css.sac.Selector. These examples are extracted from open source projects. You can vote up the examples you like and your votes will be used in our system to product more good examples.

Example 1

```
private boolean isPseudoElementRule( final
ElementStyleRule rule ) {
  final List<CSSSelector> selectorList =
rule.getSelectorList();
  for ( int i = 0; i < selectorList.size(); i +=
1 ) {
    final CSSSelector selector =
selectorList.get( i );
    if ( selector == null ) {
      continue;
    }

    if ( selector.getSelectorType() !=
Selector.SAC_CONDITIONAL_SELECTOR ) {
      continue;
    }

    final ConditionalSelector cs =
(ConditionalSelector) selector;
    final Condition condition =
cs.getCondition();
    if ( condition.getConditionType() !=
Condition.SAC_PSEUDO_CLASS_CONDITION ) {
      continue;
    }
    return true;
  }
  return false;
}
```

Example 2

```
private boolean isPseudoElementRule( final
CSSStyleRule rule ) {
  final CSSSelector selector =
rule.getSelector();
  if ( selector == null ) {
    return false;
  }

  if ( selector.getSelectorType() !=
Selector.SAC_CONDITIONAL_SELECTOR ) {
    return false;
  }

  final ConditionalSelector cs =
(ConditionalSelector) selector;
  final Condition condition = cs.getCondition();
  if ( condition.getConditionType() !=
Condition.SAC_PSEUDO_CLASS_CONDITION ) {
    return false;
  }
  return true;
}
```

Example 3

```
private static CssSelector<? extends Selector, ?
extends XPathComponent> getSelector(Selector
selector) {
        switch (selector.getSelectorType()) {
                case
Selector.SAC_CONDITIONAL_SELECTOR:
                        return
conditionalCssSelector;

                case
Selector.SAC_ELEMENT_NODE_SELECTOR:
                        return tagNameSelector;

                // COMBINATORS
                case
Selector.SAC_DESCENDANT_SELECTOR:
                        return
descendantCssSelector;
                case Selector.SAC_CHILD_SELECTOR:
                        return
directDescendantCssSelector;
```

```
                case
Selector.SAC_DIRECT_ADJACENT_SELECTOR:
                        return
directAdjacentCssSelector;
                // the parser returns this code
for the "E ~ F" selector. Go figure...
                case
Selector.SAC_ANY_NODE_SELECTOR:
                        return
generalAdjacentCssSelector;

                case
Selector.SAC_ROOT_NODE_SELECTOR:
                case
Selector.SAC_NEGATIVE_SELECTOR:
                case
Selector.SAC_TEXT_NODE_SELECTOR:
                case
Selector.SAC_CDATA_SECTION_NODE_SELECTOR:
                case
Selector.SAC_PROCESSING_INSTRUCTION_NODE_SELECTOR:
                case
Selector.SAC_COMMENT_NODE_SELECTOR:
                case
Selector.SAC_PSEUDO_ELEMENT_SELECTOR:
                default:
                        return new
UnknownCssSelector<>(selector.getSelectorType());
        }
}
```

Example 4

```
protected List querySelector(CSSRuleList
ruleList, int selectorType,
                int selectorConditionType) {
        List list = new ArrayList();
        int length = ruleList.getLength();
        for (int i = 0; i < length; i++) {
                CSSRule rule = ruleList.item(i);
                switch (rule.getType()) {
                case CSSRule.STYLE_RULE: {
                        if (rule instanceof
ExtendedCSSRule) {
                                ExtendedCSSRule r
= (ExtendedCSSRule) rule;
```

```
                                    SelectorList
selectorList = r.getSelectorList();
                                    // Loop for
SelectorList
                                    int l =
selectorList.getLength();
                                    for (int j = 0; j
< l; j++) {
                                            Selector
selector = (Selector) selectorList.item(j);
                                            if
(selector.getSelectorType() == selectorType) {

switch (selectorType) {

case Selector.SAC_CONDITIONAL_SELECTOR:

// It's conditional selector

ConditionalSelector conditionalSelector =
(ConditionalSelector) selector;

short conditionType = conditionalSelector

.getCondition().getConditionType();

if (selectorConditionType == conditionType) {

// current selector match the current CSS

// Rule

// CSSStyleRule styleRule = (CSSStyleRule)

// rule;

list.add(selector);

}
                                                    }
                                            }

                                    }
                            }
                    }
                    }
```

```
        }
        return list;
}
```

Example 5

```
protected Integer getKey(int selectorType, int
conditionType) {
        if (selectorType ==
Selector.SAC_CONDITIONAL_SELECTOR) {
                if (conditionType ==
SAC_CLASS_CONDITION.intValue())
                        return
SAC_CLASS_CONDITION;
                if (conditionType ==
SAC_ID_CONDITION.intValue())
                        return SAC_ID_CONDITION;
                if (conditionType ==
SAC_PSEUDO_CLASS_CONDITION.intValue())
                        return
SAC_PSEUDO_CLASS_CONDITION;
                return
OTHER_SAC_CONDITIONAL_SELECTOR;
        }

        return OTHER_SAC_SELECTOR;
}
```

Example 6

```
/**
 * Returns all the declarations that apply to the
given element and defined pseudo elements.
 *
 * @return The key 'null' in the returned Map
contains the node's declarations. Names keys
contain the declarations
 *         for pseudo elements.
 */
private Map<String, Map<String, LexicalUnit>>
getApplicableDeclarations(final INode node) {
        final List<PropertyDecl>
coreDeclarationsForElement =
findCoreDeclarationsFor(node);

Collections.sort(coreDeclarationsForElement,
PROPERTY_CASCADE_ORDERING);
```

```
        final List<PropertyDecl>
stylesheetDeclarationsForElement =
findAllDeclarationsFor(node);

Collections.sort(stylesheetDeclarationsForElement,
PROPERTY_CASCADE_ORDERING);

        // Both lists are sorted in cascade
order. We can then just stuff them into a map and
get the right values
        // since higher-priority values come
later and overwrite lower-priority ones.
        // Core styles are at the list's begin,
so they are ruled out by stylesheet definitions.
        final List<PropertyDecl>
rawDeclarationsForElement =
coreDeclarationsForElement;

rawDeclarationsForElement.addAll(stylesheetDeclarationsForElement);

        final Map<String, Map<String,
PropertyDecl>> distilledDeclarations = new
HashMap<String, Map<String, PropertyDecl>>();
        final Map<String, Map<String,
LexicalUnit>> values = new HashMap<String,
Map<String, LexicalUnit>>();
        // Key null for nodes direct styles
        distilledDeclarations.put(null, new
HashMap<String, PropertyDecl>());
        values.put(null, new HashMap<String,
LexicalUnit>());
        for (final PropertyDecl declaration :
rawDeclarationsForElement) {
                String pseudoElement = null;
                final Selector sel =
declaration.getRule().getSelector();
                if (sel instanceof
DescendantSelector && ((DescendantSelector)
sel).getSimpleSelector().getSelectorType() ==
Selector.SAC_PSEUDO_ELEMENT_SELECTOR) {
                        // Get the pseudo
elements name, if this declaration comes from an
SAC_PSEUDO_ELEMENT_SELECTOR
                        final ElementSelector
elementSel = (ElementSelector)
```

```
((DescendantSelector) sel).getSimpleSelector();
                        pseudoElement =
elementSel.getLocalName().toLowerCase();
                }

                PropertyDecl previousDeclaration
= null;
                if
(distilledDeclarations.containsKey(pseudoElement))
{
                        previousDeclaration =
distilledDeclarations.get(pseudoElement).get(declaration.getProperty());
                } else {

distilledDeclarations.put(pseudoElement, new
HashMap<String, PropertyDecl>());
                }
                if (previousDeclaration == null
|| !previousDeclaration.isImportant() ||
declaration.isImportant()) {

distilledDeclarations.get(pseudoElement).put(declaration.getProperty(),
declaration);
                        if
(values.containsKey(pseudoElement)) {

values.get(pseudoElement).put(declaration.getProperty(),
declaration.getValue());
                        } else {

values.put(pseudoElement, new HashMap<String,
LexicalUnit>());

values.get(pseudoElement).put(declaration.getProperty(),
declaration.getValue());
                        }
                }
        }

        return values;
}
```

Example 7

```
/**
 * Creates a pseudo element selector.
```

```
 *
 * @param pseudoName the pseudo element name.
<code>NULL</code> if this element selector can
match any pseudo
 *                   element.</p>
 * @return the element selector
 * @throws CSSException If this selector is not
supported.
 */
public ElementSelector
createPseudoElementSelector( final String
namespaceURI,

final String pseudoName )
  throws CSSException {
  return new CSSElementSelector
    ( Selector.SAC_PSEUDO_ELEMENT_SELECTOR,
namespaceURI, pseudoName );
}
```

Example 8

```
/**
 * Creates a pseudo element selector.
 *
 * @param pseudoName the pseudo element name.
<code>NULL</code> if this element selector can
match any pseudo
 *                   element.</p>
 * @return the element selector
 * @throws CSSException If this selector is not
supported.
 */
public ElementSelector
createPseudoElementSelector( String namespaceURI,

String pseudoName )
  throws CSSException {
  if ( namespaceURI == null ) {
    namespaceURI =
CSSParserContext.getContext().getDefaultNamespace();
  }
  return new CSSElementSelector
    ( Selector.SAC_PSEUDO_ELEMENT_SELECTOR,
namespaceURI, pseudoName );
}
```

Example 9

```
/**
 * {@inheritDoc}
 */
public String getCssText(final CSSFormat format)
{
    final StringBuilder sb = new StringBuilder();

    if (null != ancestorSelector_) {
        sb.append(((CSSFormatable)
ancestorSelector_).getCssText(format));
    }

    if (Selector.SAC_PSEUDO_ELEMENT_SELECTOR ==
getSimpleSelector().getSelectorType()) {
        sb.append(':');
    }
    else {
        sb.append(' ');
    }

    if (null != simpleSelector_) {
        sb.append(((CSSFormatable)
simpleSelector_).getCssText(format));
    }

    return sb.toString();
}
```

Example 10

```
/**
 * An integer indicating the type of
<code>Selector</code>
 */
public short getSelectorType() {
  if ( childRelation ) {
    return Selector.SAC_CHILD_SELECTOR;
  } else {
    return Selector.SAC_DESCENDANT_SELECTOR;
  }
}
```

Example 11

```
/**
 * An integer indicating the type of
<code>Selector</code>
 */
public short getSelectorType() {
  if ( childRelation ) {
    return Selector.SAC_CHILD_SELECTOR;
  } else {
    return Selector.SAC_DESCENDANT_SELECTOR;
  }
}
```

Example 12

```
/**
 * Creates an element selector.
 *
 * @param namespaceURI the <a
href="http://www.w3.org/TR/REC-xml-names/#dt-
NSName">namespace URI</a> of the element
 *                     selector.
 * @param tagName      the <a
href="http://www.w3.org/TR/REC-xml-names/#NT-
LocalPart">local part</a> of the element
 *                     name. <code>NULL</code> if
this element selector can match any element.</p>
 * @return the element selector
 * @throws CSSException If this selector is not
supported.
 */
public ElementSelector createElementSelector(
final String namespaceURI,

final String tagName )
  throws CSSException {
  return new CSSElementSelector
    ( Selector.SAC_ELEMENT_NODE_SELECTOR,
namespaceURI, tagName );
}
```

Example 13

```
/**
 * Creates an element selector.
 *
 * @param namespaceURI the <a
href="http://www.w3.org/TR/REC-xml-names/#dt-
```

```
NSName">namespace URI</a> of the element
 *                     selector.
 * @param tagName      the <a
href="http://www.w3.org/TR/REC-xml-names/#NT-
LocalPart">local part</a> of the element
 *                     name. <code>NULL</code> if
this element selector can match any element.</p>
 * @return the element selector
 * @throws CSSException If this selector is not
supported.
 */
public ElementSelector createElementSelector(
String namespaceURI,

String tagName )
  throws CSSException {
  if ( namespaceURI == null ) {
    namespaceURI =
CSSParserContext.getContext().getDefaultNamespace();
  }

  return new CSSElementSelector
    ( Selector.SAC_ELEMENT_NODE_SELECTOR,
namespaceURI, tagName );
}
```

Example 14

```
/**
 * Returns true if the given element matches the
given selector.
 */
private static boolean matches(final Selector
selector, final INode node) {

        if (node == null) {
                // This can happen when, e.g.,
with the rule "foo > *".
                // Since the root element matches
the "*", we check if
                // its parent matches "foo", but
of course its parent
                // is null
                return false;
        }
```

```
        final int selectorType =
selector.getSelectorType();

        switch (selectorType) {
        case Selector.SAC_CONDITIONAL_SELECTOR:
                // We end here for PseudoClass
selectors in difference to
PseudoElementSelectors.
                // See
StyleSheetReader#createParser for defined
PseudoElements.
                // according to http://www.w3.org
/TR/CSS2/selector.html#pseudo-class names are
case insensitive
                final ConditionalSelector cs =
(ConditionalSelector) selector;
                if
(cs.getCondition().getConditionType() ==
Condition.SAC_PSEUDO_CLASS_CONDITION) {
                        final AttributeCondition
ac = (AttributeCondition) cs.getCondition();
                        if (node instanceof
IComment) {
                                return
COMMENT_RULE_NAME.equalsIgnoreCase(ac.getValue())
&& matches(cs.getSimpleSelector(),
node.getParent());
                        } else {
                                return false;
                        }
                } else {
                        return
matches(cs.getSimpleSelector(), node) &&
matchesCondition(cs.getCondition(), node);
                }

        case Selector.SAC_ANY_NODE_SELECTOR:
                // You'd think we land here if we
have a * rule, but instead
                // it appears we get a
SAC_ELEMENT_NODE_SELECTOR with
                // localName==null
                return true;

        case Selector.SAC_ROOT_NODE_SELECTOR:
                return node.getParent()
```

```
instanceof IDocument;

        case Selector.SAC_NEGATIVE_SELECTOR:
                break; // not yet supported

        case Selector.SAC_ELEMENT_NODE_SELECTOR:
                final String elementName =
getLocalNameOfElement(node);
                final String selectorName =
((ElementSelector) selector).getLocalName();

                // If the Selector has a
namespace URI, it has to match the namespace URI
of the node
                final String selectorNamespaceURI
= ((ElementSelector) selector).getNamespaceURI();
                if (selectorNamespaceURI != null
&&
!selectorNamespaceURI.equals(getNamespaceURIOfElement(node)))
{
                        return false;
                }

                if (selectorName == null) {
                        // We land here if we
have a wildcard selector (*) or
                        // a pseudocondition w/o
an element name (:before)
                        // Probably other
situations too (conditional w/o element
                        // name? e.g.
[attr=value])
                        return true;
                }
                if
(selectorName.equals(elementName)) {
                        return true;
                }
                return false;

        case Selector.SAC_TEXT_NODE_SELECTOR:
                return false; // not yet
supported

        case
Selector.SAC_CDATA_SECTION_NODE_SELECTOR:
```

```
                return false; // not yet
supported

        case
Selector.SAC_PROCESSING_INSTRUCTION_NODE_SELECTOR:
                return false; // not yet
supported

        case Selector.SAC_COMMENT_NODE_SELECTOR:
                return false; // not yet
supported

        case
Selector.SAC_PSEUDO_ELEMENT_SELECTOR:
                // Always return true here. The
Selector is evaluated with SAC_CHILD_SELECTOR.
                // The pseudo element rules are
stored with the parent's rules (see
StyleSheet#getApplicableDeclarations)
                return true;

        case Selector.SAC_DESCENDANT_SELECTOR:
                final DescendantSelector ds =
(DescendantSelector) selector;
                return
matches(ds.getSimpleSelector(), node) &&
matchesAncestor(ds.getAncestorSelector(), node);

        case Selector.SAC_CHILD_SELECTOR:
                final DescendantSelector ds2 =
(DescendantSelector) selector;
                if
(ds2.getSimpleSelector().getSelectorType() ==
Selector.SAC_PSEUDO_ELEMENT_SELECTOR) {
                        return
matches(ds2.getAncestorSelector(), node);
                }
                final IParent parent =
node.getParent();
                return
matches(ds2.getSimpleSelector(), node) &&
matches(ds2.getAncestorSelector(), parent);

        case
Selector.SAC_DIRECT_ADJACENT_SELECTOR:
```

```
                final SiblingSelector ss =
(SiblingSelector) selector;

                if (node != null &&
node.getParent() != null &&
matches(ss.getSiblingSelector(), node)) {

                        // find next sibling
                        final Iterator<INode> i =
node.getParent().children().iterator();
                        INode e = null;
                        INode f = null;
                        while (i.hasNext() && e
!= node) {
                                f = e;
                                e = i.next();
                        }

                        if (e == node) {
                                return
matches(ss.getSelector(), f);
                        }
                }
                return false;

        default:
                // System.out.println("DEBUG:
selector type not supported");
                // TODO: warning: selector not
supported
        }
        return false;
}
```

Example 15

```
/**
 * Returns true if the given element matches the
given selector.
 */
private static boolean matches(Selector selector,
VEXElement element) {

        if (element == null) {
                // This can happen when, e.g.,
with the rule "foo > *".
```

```
                // Since the root element matches
the "*", we check if
                // its parent matches "foo", but
of course its parent
                // is null
                return false;
        }

        String elementName = element.getName();
        int selectorType =
selector.getSelectorType();

        switch (selectorType) {

        case Selector.SAC_ANY_NODE_SELECTOR:
                // You'd think we land here if we
have a * rule, but instead
                // it appears we get a
SAC_ELEMENT_NODE_SELECTOR with
                // localName==null
                return true;

        case Selector.SAC_CONDITIONAL_SELECTOR:
                // This little wart is the
product of a mismatch btn the CSS
                // spec an the Flute parser. CSS
treats pseudo-elements as elements
                // attached to their parents,
while Flute treats them like
                // attributes
                ConditionalSelector cs =
(ConditionalSelector) selector;
                if
(cs.getCondition().getConditionType() ==
Condition.SAC_PSEUDO_CLASS_CONDITION) {
                        if (element instanceof
PseudoElement) {

AttributeCondition ac = (AttributeCondition) cs

.getCondition();
                                return
ac.getValue().equals(element.getName())

&& matches(cs.getSimpleSelector(), element
```

```
.getParent());
                        } else {
                                return false;
                        }
                } else {
                        return
matches(cs.getSimpleSelector(), element)
                                                &&
matchesCondition(cs.getCondition(), element);
                }

        case Selector.SAC_ELEMENT_NODE_SELECTOR:
                String selectorName =
((ElementSelector) selector).getLocalName();
                if (selectorName == null) {
                        // We land here if we
have a wildcard selector (*) or
                        // a pseudocondition w/o
an element name (:before)
                        // Probably other
situations too (conditional w/o element
                        // name? e.g.
[attr=value])
                        return true;
                }
                if
(selectorName.equals(elementName)) {
                        return true;
                }
                break;

        case Selector.SAC_DESCENDANT_SELECTOR:
                DescendantSelector ds =
(DescendantSelector) selector;
                return
matches(ds.getSimpleSelector(), element)
                                &&
matchesAncestor(ds.getAncestorSelector(), element

.getParent());

        case Selector.SAC_CHILD_SELECTOR:
                DescendantSelector ds2 =
(DescendantSelector) selector;
                VEXElement parent =
element.getParent();
```

```
                if (element instanceof
PseudoElement) {
                        parent =
parent.getParent(); // sigh - this looks
inelegant, but

// whatcha gonna do?
                }
                return
matches(ds2.getSimpleSelector(), element)
                                &&
matches(ds2.getAncestorSelector(), parent);

        case
Selector.SAC_DIRECT_ADJACENT_SELECTOR:

                SiblingSelector ss =
(SiblingSelector) selector;

                if (element != null &&
element.getParent() != null
                                &&
matches(ss.getSiblingSelector(), element)) {

                        // find next sibling
                        final
Iterator<VEXElement> i =
element.getParent().getChildElements().iterator();
                        VEXElement e = null;
                        VEXElement f = null;
                        while (i.hasNext() && e
!= element) {
                                f = e;
                                e = i.next();
                        }

                        if (e == element) {
                                return
matches(ss.getSelector(), f);
                        }
                }
                return false;

        default:
                // System.out.println("DEBUG:
selector type not supported");
```

```
                // TODO: warning: selector not
supported
        }
        return false;
}
```

Example 16

```
/**
 * Returns true if the given element matches the
given selector.
 */
private static boolean matches(Selector selector,
Element element) {

    if (element == null) {
        // This can happen when, e.g., with the
rule "foo > *".
        // Since the root element matches the
"*", we check if
        // its parent matches "foo", but of
course its parent
        // is null
        return false;
    }

    String elementName = element.getName();
    int selectorType =
selector.getSelectorType();

    switch (selectorType) {

    case Selector.SAC_ANY_NODE_SELECTOR:
        // You'd think we land here if we have a
* rule, but instead
        // it appears we get a
SAC_ELEMENT_NODE_SELECTOR with localName==null
        return true;

    case Selector.SAC_CONDITIONAL_SELECTOR:
        // This little wart is the product of a
mismatch btn the CSS
        // spec an the Flute parser. CSS treats
pseudo-elements as elements
        // attached to their parents, while Flute
treats them like attributes
```

```
        ConditionalSelector cs =
(ConditionalSelector) selector;
        if (cs.getCondition().getConditionType()
== Condition.SAC_PSEUDO_CLASS_CONDITION) {
            if (element instanceof PseudoElement)
{
                AttributeCondition ac =
(AttributeCondition) cs.getCondition();
                return
ac.getValue().equals(element.getName())
                    &&
matches(cs.getSimpleSelector(),
element.getParent());
            } else {
                return false;
            }
        } else {
            return
matches(cs.getSimpleSelector(), element) &&

matchesCondition(cs.getCondition(), element);
        }

    case Selector.SAC_ELEMENT_NODE_SELECTOR:
        String selectorName =
((ElementSelector)selector).getLocalName();
        if (selectorName == null) {
            // We land here if we have a wildcard
selector (*) or
            // a pseudocondition w/o an element
name (:before)
            // Probably other situations too
(conditional w/o element
            // name? e.g. [attr=value])
            return true;
        }
        if (selectorName.equals(elementName)) {
            return true;
        }
        break;

    case Selector.SAC_DESCENDANT_SELECTOR:
        DescendantSelector ds =
(DescendantSelector) selector;
        return matches(ds.getSimpleSelector(),
element) &&
```

```
matchesAncestor(ds.getAncestorSelector(),
element.getParent());

    case Selector.SAC_CHILD_SELECTOR:
        DescendantSelector ds2 =
(DescendantSelector) selector;
        Element parent = element.getParent();
        if (element instanceof PseudoElement) {
            parent = parent.getParent(); // sigh
- this looks inelegant, but whatcha gonna do?
        }
        return matches(ds2.getSimpleSelector(),
element) &&
            matches(ds2.getAncestorSelector(),
parent);

    case Selector.SAC_DIRECT_ADJACENT_SELECTOR:

        SiblingSelector ss = (SiblingSelector)
selector;

        if (element != null &&
element.getParent() != null
                &&
matches(ss.getSiblingSelector(), element) ) {

            // find next sibling

            final Iterator i =
element.getParent().getChildIterator();
            Element e = null;
            Element f = null;

            while (i.hasNext() && e != element) {
                f = e;
                e = (Element)i.next();
            }

            if (e == element) {
                return matches(ss.getSelector(),
f);
            }
        }
        return false;
```

```
        default:
            //System.out.println("DEBUG: selector
    type not supported");
            // TODO: warning: selector not supported
        }
        return false;
    }
```