



Clipcode Snippets for W3C Web DOM Core

Specification

<http://www.w3.org/TR/domcore/>

Overview

The Web DOM Core is an updated (compatible) description of how the DOM (Document Object Model) should work. To quote from the specification:

The goal of this specification is taking all of DOM Level 3 Core and the "DOM Event Architecture" and "Basic Event Interfaces" chapters of DOM Level 3 Events .. and do the following:

Align them with the needs of ECMAScript where possible.

Align them with existing implementations.

Simplify them as much as possible.

Import bits of HTML5 that ought to be in DOM Core.

Prevent a dependency on HTML5."

To help web developers use Web DOM Core, Clipcode has provides a collection of snippets showing use of its features.

Table of Contents

Attr.....	2
CharacterData.....	3
Document.....	4
DocumentType.....	7
DOMException.....	8
DOMImplementation.....	9
DOMSettableTokenList.....	10
DOMTokenList.....	11
Element.....	12
HtmlCollection.....	14
Node.....	15
NodeList.....	18
Text.....	19

Attr

```
<!DOCTYPE html>
<html>
<head>
<title>Attr Snippet</title>
</head>
<body>
<h1>The Attr Interface</h1>
<div id="root">
<p>This snippet demonstrates use of Attr, which represents an attribute of an element.</p>
  <div id="parent">
    <p>Sample Content</p>
    <input id="a1" name="first" type="button" value="a1" />
  </div>
</div>
<script>
  try {
    var firstAttr = document.getElementById("a1").attributes[0];

    document.write("<h2>Attr Attributes</h2>");
    document.write("<p>namespaceURI = " + firstAttr.namespaceURI);
    document.write("<p>prefix = " + firstAttr.prefix);
    document.write("<p>localName = " + firstAttr.localName);
    document.write("<p>tagName = " + firstAttr.name);
    document.write("<p>value = " + firstAttr.value);
    firstAttr.value = "aaaa";
    document.write("<p>[after setting value] value = " + firstAttr.value);

  } catch (ex) {
    alert(ex);
  }
</script>
</body>
</html>
```

CharacterData

```

<!DOCTYPE html>
<html>
<head>
<title>CharacterData Snippet</title>
</head>
<body>
<h1>The CharacterData Interface</h1>
<div id="root">
<p>This snippet demonstrates use of CharacterData.</p>
<p>Note Character Data is not the same as CDATASection (CDATA tag). </p>
  <div id="parent">
    <p id="myPara">Sample Content</p>
  </div>
</div><script>
  try {
    var myPara = document.getElementById("myPara");
    // note Paragraph.children returns the children that are elements (in this case, 0).
    // note Paragraph.firstChild returns the first child node, which does not necessarily have to be
    an element.

    // In this case, the firstChild is a Text node.
    // Text implements CharacterData interface.
    var myCharacterData = myPara.firstChild;
    document.write("<p>data = " + myCharacterData.data);
    document.write("<p>length = " + myCharacterData.length);
    document.write("<p>substringData(2,4) = " + myCharacterData.substringData(2, 4));
    myCharacterData.appendData("ABC");
    myCharacterData.insertData(6,"ABC");
    myCharacterData.deleteData(10,2);
    myCharacterData.replaceData(1,4,"ABC");
  } catch (ex) {
    alert(ex);
  }
</script></body> </html>

```

Document

```

<!DOCTYPE html>
<html>
<head>
<title>Document Snippet</title>
</head>
<body>
<h1>The Document Interface</h1>
<div id="root">
<p>This snippet demonstrates use of Document (derived from Node), which represents a document
</p>
  <div id="parent" class="important">
    <p>Sample Content</p>
    <input id="a1" name="first" type="button" value="a1" />
  </div>
</div>

<script>

  try {
    // Note that Web DOM Core's Document interface does not have a method called write, but
    HTML5's HtmlDocument interface does.
    // So when we use document.write, this is from HtmlDocument.
    // Contrary to popular belief, HtmlDocument does not derive from Document!
    // Instead, section 3.1.1 of Html5 spec states that all Document objects must also impleemnt
    the HtmlDocument interface

    // document is a member of the window global

    document.write(document); // writes out object HtmlDocument

    var myDOMImpl = document.implementation;
    document.write(myDOMImpl);

    document.write("<p>documentURI = " + document.documentURI);
  }

```

```
document.write("<p>compatMode = " + document.compatMode);

var myDocType = document.doctype;
document.write(myDocType);

// var myDocElement = document.documentElement;
// if (myDocElement != null)
//   document.write(documentElement);

// gets
var myNodeList = document.getElementsByTagName("*");
document.write("<p>getElementsByTagName('*') returned " + myNodeList.length + "
entries");
myNodeList = document.getElementsByTagName("div");
document.write("<p>getElementsByTagName('div') returned " + myNodeList.length + "
entries");

var myNodeList1 = document.getElementsByTagNameNS("http://www.w3.org/1999/xhtml",
"div"); // IMPORTANT: namespace should NOT end in '/'
document.write("<p>getElementsByTagNameNS('http://www.w3.org/1999/xhtml', 'div')
returned " + myNodeList1.length + " entries");

var myNodeList2 = document.getElementsByClassName("important");
document.write("<p>getElementsByClassName('important') returned " + myNodeList2.length
+ " entries");

var myElement = document.getElementById("parent");
if (myElement != null)
  document.write("<p>Element 'parent' found");

// creates
var newHR = document.createElement("hr");
myElement.appendChild(newHR);
var newHR = document.createElementNS("http://www.w3.org/1999/xhtml", "hr");
myElement.appendChild(newHR);

var newFrag = document.createDocumentFragment();
```

```

var newText = document.createTextNode("TEXT HERE");
myElement.appendChild(newText);
var newComment = document.createComment("COMMENT HERE");
myElement.appendChild(newComment);
var newPI = document.createProcessingInstruction("xyz");
var newFrag = document.createDocumentFragment();
} catch (ex) {
    alert(ex);
}
</script>
<h2>Sample Frame</h2>
<div>
    <iframe id="myFrame" src="DocumentFramed.html" width="600" height="80"></iframe> <!--
NOTE: IFRAME end-tag required -->
    <p>Finished</p>
</div>
<script>
/* try {
    // import and adopt node
    var myFrame = document.getElementById("myFrame");
    alert(myFrame.contentDocument);
    var myElementFromFrame = myFrame.contentDocument.getElementsByClassName("abc");
    if (myElementFromFrame == null)
        alert("error finding myElementFromFrame");
    else {
        alert(myElementFromFrame.length);
        var myImportedElement = document.importNode(myElementFromFrame, true);
        alert(myImportedElement);
        myElement.appendChild(myImportedElement);
    }
} catch (ex) {
    alert(ex);
}
*/
</script></body></html>

```

DocumentType

```
<!DOCTYPE html>
<html>
<head>
<title>DocumentType Snippet</title>
</head>
<body>
<h1>The DocumentType Interface</h1>
<div id="root">
<p>This snippet demonstrates use of DocumentType.</p>
  <div id="parent">
    <p>Sample Content</p>
    <input id="a1" name="first" type="button" value="a1" />
  </div>
</div>
<script>

  try {
    var myDocType = document.doctype; // note attribute is caled docType (with lower 't'), not
    DocumentType or docType.

    document.write("<h2>DocumentType Attributes</h2>");
    document.write("<p>name = " + myDocType.name);
    document.write("<p>publicId= " + myDocType.publicId);
    document.write("<p>systemId = " + myDocType.systemId);

  } catch (ex) {
    alert(ex);
  }
</script>
</body>
</html>
```

DOMException

```

<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>The DOMException Interface</h1>
<div id="root">
<p>This snippet demonstrates use of DOMException by getting the browser to generate a sample
exception and we catch it. </p>
  <div id="goodElement">
    <p>hello3</p>
  </div>
</div>
<p>
<script>
try {
  // Find element we wish to replace in the tree
  var existingElement = document.getElementById("badElement"); // Change to "goodElement" to
see code working

  if (existingElement === undefined)
    document.write("existingElement NOT FOUND");
    // at this point, we should be taking a different path upon detecting the error,
    // but for this snippet, let's continue

  // Create a new element
  var newElement = document.createElement("newElement"); // to generate an Error exception,
comment this line out
  newElement.innerHTML = "New element goes here";

  // insert the new element
  var root= document.getElementById("root");
  root.replaceChild(newElement, existingElement); // replaceChild must be called on immediate
parent of existingElement, not document

if (ex instanceof DOMException) {
  alert("Is DOMException");
  alert("DOMException.code = " + ex.code.toString());
  alert("DOMException.name = " + ex.name);
} else if (ex instanceof Error) {
  alert("Is Error");
  alert("ex.message = " + ex.message);
}
// for (var x in ex) alert(x); // this would show all properties of exception
}
</script></html>

```

DOMImplementation

```
<!DOCTYPE html>
<html>
<head>
<title>DOMImplementation Snippet</title>
</head>
<body>
<h1>The DOMImplementation Interface</h1>
<div id="root">
<p>This snippet demonstrates use of DOMImplementation, which provides access to the
implementation of the DOM, to create new documents and inquire about features</p>
  <div id="parent">
    <p>Sample Content</p>

  </div>
</div>
<script>

  try {
    myFeature =
document.implementation.hasFeature("http://www.w3.org/TR/SVG11/feature#SVG", "1.1");
    alert("[Testing valid feature/version] hasFeature('http://www.w3.org/TR/SVG11/feature#SVG',
'1.1') returned = " + myFeature);

    myFeature =
document.implementation.hasFeature("http://www.w3.org/TR/SVG11/feature#SVG", "2.0");
    alert("[Testing later version] hasFeature('http://www.w3.org/TR/SVG11/feature#SVG', '2.0')
returned = " + myFeature);

    var myFeature = document.implementation.hasFeature("xxx", "yyy");
    alert("[Testing non-existent feature] hasFeature(yyy) returned = " + myFeature);

    var myDocType = document.implementation.createDocumentType("qualifiedName", "xxx",
"yyy");
    var myDoc = document.implementation.createDocument("http://example.com", "myName",
myDocType);
    var myHtmlDoc = document.implementation.createHTMLDocument("MY TITLE");

  } catch (ex) {
    alert(ex);
  }

</script>
</body>
</html>
```

DOMSettableTokenList

```

<!DOCTYPE html>
<html>
<head>
<title>DOMSettableTokenList Snippet</title>
<link id="a1" rel="icon" href="myIcons" sizes="16x16" >
</head>
<body>
<h1>The DOMSettableTokenList Interface</h1>
<div id="root">
<p>This snippet demonstrates use of DOMSettableTokenList, which is a settable list of tokens
stored as a space-separated string</p>
</div>
<script>

    try {
        document.write("<h2>Accessing a DOMSettableTokenList via
HtmlLinkElement.sizes</h2>");
        var myLink = document.getElementById("a1");
        if (myLink == null)
            document.write("<p>link not found");
        else {
            for (var x in myLink) alert(x);

            var myDOMSettableTokenList = myLink.sizes;
            if (myDOMSettableTokenList == undefined)
                document.write("<p>myLink.sizes undefined");
            else {

                document.write("<p>Number of entries in DOMSettableTokenList = " +
myDOMSettableTokenList.length);

                document.write("<p>Result of accessing tokens via
DOMSettableTokenList.item(index)");
                for (var x = 0; x < myDOMSettableTokenList.length; x++)
                    document.write("<p> * " + myDOMSettableTokenList.item(x));
            }
        }
    } catch (ex) {
        alert(ex);
    }

</script>
</body>
</html>

```

DOMTokenList

```

<!DOCTYPE html>
<html><head>
<title>DOMTokenList Snippet</title>
</head><body>
<h1>The DOMTokenList Interface</h1>
<div id="root">
<p>This snippet demonstrates use of DOMTokenList, which is a list of tokens stored as a space-
separated string</p>
  <p>Sample Content</p>
  <div id="parent">
    <input id="a1" type="button" class="a b c"/>First input <br />
  </div></div>
<script>
  try {
    document.write("<h2>Accessing a DOMTokenList via HtmlElement.classList</h2>");
    var myInput = document.getElementById("a1");
    var myDOMTokenList = myInput.classList;
    document.write("<p>Number of entries in DOMTokenList = " + myDOMTokenList.length);

    document.write("<p>Result of accessing tokens via DOMTokenList.item(index)");
    for (var x = 0; x < myDOMTokenList.length; x++)
      document.write("<p> * " + myDOMTokenList.item(x));

    document.write("<p>Checking if it contains 'a'");
    if (myDOMTokenList.contains("a"))
      document.write("<p>Contains 'a'");
    else
      document.write("<p>Does not contain 'a'");
    document.write("<p>Result of adding a 'd' token via DOMTokenList.add(DOMString)");
    myDOMTokenList.add("d");
    for (var x = 0; x < myDOMTokenList.length; x++)
      document.write("<p> * " + myDOMTokenList.item(x));
    document.write("<p>Result of removing the 'b' token via
DOMTokenList.remove(DOMString)");
    myDOMTokenList.remove("b");
    for (var x = 0; x < myDOMTokenList.length; x++)
      document.write("<p> * " + myDOMTokenList.item(x));
    document.write("<p>Result of toggling 'a' (already exists) and 'e' does not exist via
DOMTokenList.toggle(DOMString)");
    myDOMTokenList.toggle("a");
    myDOMTokenList.toggle("e");
    for (var x = 0; x < myDOMTokenList.length; x++)
      document.write("<p> * " + myDOMTokenList.item(x));
  } catch (ex) {
    alert(ex);
  }
</script></body></html>

```

Element

```

<!DOCTYPE html>
<html>
<head>
<title>Element Snippet</title>
</head>
<body>
<h1>The Element Interface</h1>
<div id="root">
<p>This snippet demonstrates use of Element, which represents an element in the document
tree.</p>
  <div id="parent">
    <p>Sample Content</p>
    <input id="a1" name="first" type="button" value="a1" />
    <input id="a2" name="second" class="abc" type="button" value="a2" />
  </div>
</div>
<script>

```

```

  try {
    var firstElement = document.getElementById("a1");

    document.write("<h2>Element Attributes</h2>");
    document.write("<p>namespaceURI = " + firstElement.namespaceURI);
    document.write("<p>prefix = " + firstElement.prefix);
    document.write("<p>localName = " + firstElement.localName);
    document.write("<p>tagName = " + firstElement.tagName);

    document.write("<p><p>Attributes");
    var attrList = firstElement.attributes;
    for (var x = 0; x < attrList.length; x++)
      document.write("<p> * " + attrList[x].name + " = " + attrList[x].value + " (" +
attrList[x].namespaceURI + ")");

```

// Note that attributes returns the attributes (i.e. attribute objects) of the element, whereas
getAttribute returns just the value (DOMString)

```

    var myAttrValue = firstElement.getAttribute("id");
    document.write("<p>getAttribute('id') returns: " + myAttrValue);
    document.write("<p>getAttribute('id') returns: " + myAttrValue);
    myAttrValue = firstElement.getAttributeNS(null, "id");
    document.write("<p>getAttributeNS('id') returns: " + myAttrValue);

    firstElement.setAttribute("value", "xyz");
    firstElement.setAttributeNS(null, "type", "checkbox");

    var secondElement = document.getElementById("a2");
    if (secondElement.hasAttribute("value"))

```

```
document.write("<p>[BEFORE REMOVE]SecondElement has value attribute");

secondElement.removeAttribute("value");
secondElement.removeAttributeNS(null, "value");

if (!(secondElement.hasAttributeNS(null, "value")))
    document.write("<p>[AFTER REMOVE]SecondElement does not have value attribute");

var myChildren = secondElement.children;
if (myChildren.length == 0)
    document.write("<p>secondElement has no children");

var parentElement = document.getElementById("parent");
myChildren = parentElement.children;
document.write("<p>Number of children of parent = " + myChildren.length);

var myChildInputElement = parentElement.getElementsByTagName("input");
document.write("<p>Number of matching elements = " + myChildInputElement.length);

myChildInputElement = parentElement.getElementsByTagNameNS(null, "input");
document.write("<p>Number of matching elements = " + myChildInputElement.length);

myChildInputElement = parentElement.getElementsByClassName("abc");
document.write("<p>Number of matching class elements = " +
myChildInputElement.length);

    } catch (ex) {
        alert(ex);
    }

</script>
</body>
</html>
```

HtmlCollection

```

<!DOCTYPE html>
<html><head>
<title>HtmlCollection Snippet</title>
</head>
<body>
<h1>The HtmlCollection Interface</h1>
<div id="root">
<p>This snippet demonstrates use of HtmlCollection, which is a collection of Html elements </p>
  <p>Sample Content</p>
  <div id="parent">
    <input id="a1" type="button" name="first" />First input <br />
    <input id="a2" type="checkbox" name="second"/>Second Input <br />
    <input id="a3" type="radio" name="third" />Third Input <br />
  </div>
</div>
<p>
</div>
<script>
  try {
    document.write("<h2>Accessing an HtmlCollection of sample content via
node.children</h2>");
    var myParent = document.getElementById("parent");
    var myHtmlCollection = myParent.children;
    document.write("<p>Number of elements in HtmlCollection = " + myHtmlCollection.length);
    document.write("<p>Result of accessing elements via myHtmlCollection.item(index)");
    for (var x = 0; x < myHtmlCollection.length; x++)
      document.write("<p> * " + myHtmlCollection.item(x).id);
    document.write("<p>Result of accessing a named element via
myHtmlCollection.namedItem(name)");
    document.write("<p> * " + myHtmlCollection.item("a2").name); // expect "second"
  } catch (ex) {
    alert(ex);
  }
</script></body></html>

```

Node

```

<!DOCTYPE html>
<html>
<head>
<title>Node Snippet</title>
</head>
<body>
<h1>The Node Interface</h1>
<div id="root">
  <p>This snippet demonstrates use of Node, which is the base for elements, etc.</p>
  <div id="parent">
    <p>Sample <b>Content</b> here</p>
  </div>
</div>
<script>

  try {
    document.write("<h2>Accessing a Node via document.getElementById</h2>");
    var myNode = document.getElementById("parent");
    document.write("<p>nodeType = " + myNode.nodeType);
    if (myNode.nodeType == Node.ELEMENT_NODE) // which it is
      document.write("<p>nodeType == ELEMENT_NODE");

    document.write("<p>nodeName = " + myNode.nodeName); // DIV, which is the tagName
    (converted to upper-case)

    document.write("<p>baseURI = " + myNode.baseURI);

    document.write("<p>ownerDocument = " + myNode.ownerDocument);
    if (document == myNode.ownerDocument) // which it is
      document.write("<p>ownerDocument returns current document");

    var myRootNode = document.getElementById("root");
    if (myRootNode == myNode.parentNode)

```

```
document.write("<p>parentNode has the name 'root' ");

if (myRootNode == myNode.parentNode)
    document.write("<p>parentNode has the name 'root' ");

if (myNode.hasChildNodes() == true)
    document.write("<p>hasChildNodes is true ");

// don't forget, hasChildNodes is a method (requires '()'), whereas childNodes is an attribute (no
'()')
var myChildNodes = myNode.childNodes;

for (var x = 0; x < myChildNodes.length; x++)
    document.write("<p> * " + myChildNodes.item(x));

var myFirstChild = myNode.firstChild;
document.write("<p>First child = " + myFirstChild.nodeType);
if (myFirstChild.nodeType == Node.TEXT_NODE)
    document.write("<p>First child's type is = TEXT_NODE");

var myLastChild = myNode.lastChild;
document.write("<p>Last child = " + myLastChild.nodeType);

var myPreviousSibling = myNode.previousSibling;
document.write("<p>Previous sibling = " + myPreviousSibling.nodeType);

var myNextSibling = myNode.nextSibling;
document.write("<p>Next sibling = " + myNextSibling.nodeType);

var result = myNode.compareDocumentPosition(myRootNode);
if (result & Node.DOCUMENT_POSITION_PRECEDING)
    document.write("<p>current node is preceded by root node");
if (result & Node.DOCUMENT_POSITION_CONTAINS)
    document.write("<p>current node is contained by root node");
if (!(result & Node.DOCUMENT_POSITION_CONTAINED_BY)) // Note negative test
```

```
document.write("<p>current node does NOT contain the root node");

document.write("<p>nodeValue = " + myNode.nodeValue); // null
document.write("<p>textContent = " + myNode.textContent); // 'sample content here'

var newChild = document.createTextNode("THIS IS INSERTED TEXT y");
myRootNode.insertBefore(newChild, myNode);

var newChild2 = document.createTextNode("THIS IS MORE INSERTED TEXT ");
myRootNode.appendChild(newChild2, myNode);

myRootNode.removeChild(newChild2);

var newChild3 = document.createTextNode("THIS IS MORE INSERTED TEXT AGAIN");
myRootNode.replaceChild(newChild3, newChild);

var myClonedNode = myNode.cloneNode(true);
document.write("<p>Cloned node's NodeType = " + myClonedNode.nodeType);

if (!myNode.isSameNode(myClonedNode)) // note negative
    document.write("<p>Cloned node is not the same as the current node");

if (myNode.isEqualNode(myClonedNode))
    document.write("<p>Cloned node is equal to the current node");

document.write("<p>lookupPrefix = " +
myNode.lookupPrefix("http://www.w3.org/XML/1998/namespace"));
document.write("<p>lookupNamespaceURI = " + myNode.lookupNamespaceURI("xmlns"));
if (myNode.isDefaultNamespace("http://www.w3.org/1999/xhtml") == true) // true
    document.write("<p>isDefaultNamespace = true");
} catch (ex) {
    alert(ex);
}
</script></body></html>
```

NodeList

```
<!DOCTYPE html>
<html>
<head>
<title>NodeList Snippet</title>
</head>
<body>
<h1>The NodeList Interface</h1>
<div id="root">
<p>This snippet demonstrates use of NodeList, which is a collection of nodes (usually in tree order)
</p>
<div id="parent">
  <p>Sample Content</p>
  <input id="a1" name="first" type="button" value="a1" />
  <input id="a2" name="first" type="checkbox" />a2
  <input id="a3" name="first" type="radio"/>a3
</div>
</div>
<script>
  try {
    document.write("<h2>Accessing a NodeList of sample content via
document.getElementsByName</h2>");
    var myNodeList = document.getElementsByName("first");
    document.write("<p>Number of nodes in node list = " + myNodeList.length);
    document.write("<p>Result of accessing nodes via NodeList.item");
    for (var x = 0; x < myNodeList.length; x++)
      document.write("<p> * " + myNodeList.item(x).id);

  } catch (ex) {
    alert(ex);
  }
</script>
</body>
</html>
```

Text

```
<!DOCTYPE html>
<html>
<head>
<title>Text Snippet</title>
</head>
<body>
<h1>The Text Interface</h1>
<div id="root">
<p>This snippet demonstrates use of Text.</p>
  <div id="parent">
    <p id="myPara">Sample Content</p>
  </div>
</div>
<script>

  try {
    var myPara = document.getElementById("myPara");
    var myText = myPara.firstChild;

    document.write("<p>wholeText = " + myText.wholeText);
    var myNewText = myText.splitText(5);
    var myReplacedText = myText.replaceWholeText("XYZ");
  } catch (ex) {
    alert(ex);
  }

</script>
</body>
</html>
```