

Document Object Model (DOM)

A brief introduction

Overview of DOM

Document Object Model (DOM) is a collection of platform-independent abstract data types (interfaces) for access and manipulation of documents.

There exist several levels of DOM, which identify interfaces for various specific functionality. At each level, there is a *core* specification, providing basic functionality and extensions, providing extra interfaces for specific tasks/document types. For example DOM **Level 1** specification includes DOM for HTML. A short overview of the DOM levels follows.

DOM Level 0. Historically, DOM was developed as a response to the need to standardize the internal HTML document model for world wide web browsers. As such **Level 0** functionality summarizes legacy interfaces, which predated the establishment of DOM. In **DOM Level 0**, a document is represented as a collection of lists (tables, arrays) of various components that it is formed of (e.g., for HTML documents: anchor tags, images, etc).

DOM Level 1. Documents are represented as tree structures (DOM Trees). Different types of nodes are defined, and DOM Tree traversal APIs are given for each type. The APIs are *atomic* - traversal is performed one edge at a time, but powerful - any algorithm for working with trees can be built on top of these APIs.

DOM Level 2. Extends **DOM Level 1** with support for *namespaces*¹ It also extends DOM core with new APIs for views, events and stylesheet support.

DOM Level 3. Further extension of **DOM Level 2** with refined namespace support, as well as support for loading and saving documents and for their validation.

¹Namespaces is a rather broad concept. In a nutshell, a namespace is a way to bind an XML element to a specific markup spec, and allow for the use of XML elements from different markup specs to encode information... In reality, things are more complicated.

DOM Level 1 (Core)

We will concentrate on functionality provided in DOM **Level 1**. DOM is a W3C standard. W3C activity related to DOM is documented at

<http://www.w3.org/DOM/>

The DOM **Level 1** recommendation can be found at

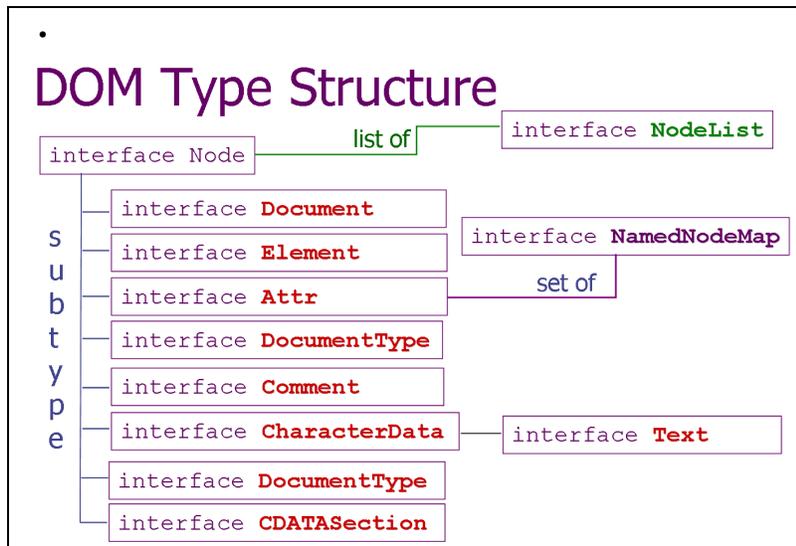
<http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>

(note: this is a pointer to the W3C recommendation. There is a working draft of second edition of this document as well.).

DOM **Level 1** (core) consists of a number of interfaces (ADTs) designed to represent an entire XML document, its individual components, and collections of these components. This handout provides a brief overview of these interfaces. Some information is omitted for clarity, refer to the original Recommendation for the full spec.

DOM Level 1 (Core) Interface structure

The structure of the main part of DOM **Level 1** is shown on the diagram below.



DOM **Level 1** defines one interface, `interface Node` to represent any component of the document. This interface is further refined to create special-purpose interfaces of the following types:

<code>interface Document:</code>	representation of the entire document
<code>interface Element:</code>	XML element nodes
<code>interface Attr:</code>	XML attribute nodes
<code>interface DocumentType:</code>	DTD nodes
<code>interface Comment:</code>	comments
<code>interface CharacterData:</code>	any parsed character data in the document
<code>interface Text : CharacterData:</code>	text nodes
<code>interface CDATASection:</code>	nodes for CDATA sections

Each interface contains a list of fields (attributes) and a list of methods. Brief overview of each interface is provided below.

General Notes

DOM **Level 1** assumes the following abstraction of an XML document. The entire document is accessed via the `Document` interface. DOM implementations must create class `Document` to represent XML documents.

From the `Document` object, access is granted to its components. The components and their nesting are determined according to the XML recommendation. E.g., the `Document` object can contain several `Comment` objects, but only one `Element` object.

All these abstract types are subtypes of a single type `Node` which provides basic access and traversal functionality. Individual subtypes tailor the API functionality to suit their roles.

Interface Node

The formal definition of this interface is reproduced from the Recommendation:

```
interface Node {

    readonly attribute DOMString      nodeName;
    attribute DOMString      nodeValue;
    readonly attribute unsigned short nodeType;
    readonly attribute Node        parentNode;
    readonly attribute NodeList    childNodes;
    readonly attribute Node        firstChild;
    readonly attribute Node        lastChild;
    readonly attribute Node        previousSibling;
    readonly attribute Node        nextSibling;
    readonly attribute NamedNodeMap attributes;
    readonly attribute Document    ownerDocument;

    Node        insertBefore(in Node newChild,
                            in Node refChild)
                            raises(DOMException);
    Node        replaceChild(in Node newChild,
                             in Node oldChild)
                             raises(DOMException);
    Node        removeChild(in Node oldChild)
                            raises(DOMException);
    Node        appendChild(in Node newChild)
                            raises(DOMException);
    boolean     hasChildNodes();
    Node        cloneNode(in boolean deep);
};
```

The brief explanation of the fields defined in this interface is below:

Type	Attribute	Explanation
DOMString	nodeName	name of the node
DOMString	nodeValue	the value of the node (see below)
unsigned short	nodeType	type of the node (see table below)
Node	parentNode	parent of the node in the DOM tree
NodeList	childNodes	List of all child nodes of the node in the DOM tree
Node	firstChild	first child of the node in the DOM tree
Node	lastChild	last child of the node in the DOM tree
Node	previousSibling	previous sibling of the node in the DOM tree
Node	nextSibling	next sibling of the node in the DOM tree
NamedNodeMap	attributes	Attributes of the node
Document	ownerDocument	document object to which this node belongs

Here are the methods:

Return Type	Method	Explanation
Node	insertBefore(newChild, refChild)	inserts newChild before the refChild
Node	replaceChild(newChild, oldChild)	replaces oldChild with newChild
Node	removeChild(in Node oldChild)	removes oldChild from the list of children nodes
Node	appendChild(in Node newChild)	adds newChild as the last child of the node
boolean	hasChildNodes()	checks if the node has children in the DOM tree
Node	cloneNode(in boolean deep)	creates a deep copy of the node

The `nodeType` attribute of the `Node` object contains information about the type of the attribute. The types are defined as follows.

Node type constant	Value	Node type
ELEMENT_NODE	1	XML Element
ATTRIBUTE_NODE	2	XML Attribute
TEXT_NODE	3	Text (leaf node)
CDATA_SECTION_NODE	4	CDATA Section
ENTITY_REFERENCE_NODE	5	Entity Reference
ENTITY_NODE	6	Entity
PROCESSING_INSTRUCTION_NODE	7	Processing instruction
COMMENT_NODE	8	Comment
DOCUMENT_NODE	9	Entire XML document
DOCUMENT_TYPE_NODE	10	DTD description
DOCUMENT_FRAGMENT_NODE	11	document fragment
NOTATION_NODE	12	notation

Depending on the *type* of the node, `nodeName`, `nodeValue` and `attributes` acquire different purposes. The following table summarizes the possible combinations.

nodeType	type of node	nodeName	nodeValue	attributes
1	Element	tag name	<i>null</i>	NamedNodeMap
2	Attr	name of attribute	value of attribute	<i>null</i>
3	Text	"#text"	content	<i>null</i>
4	CDATASection	"#cdata-section"	content	<i>null</i>
5	EntityReference	name of entity referenced	<i>null</i>	<i>null</i>
6	Entity	entity name	<i>null</i>	<i>null</i>
7	ProcessingInstruction	target	content – target	<i>null</i>
8	Comment	"#comment"	content	<i>null</i>
9	Document	"#document"	<i>null</i>	<i>null</i>
10	DocumentType	document type name	<i>null</i>	<i>null</i>
11	DocumentFragment	"#document-fragment"	<i>null</i>	<i>null</i>
12	Notation	notation name	<i>null</i>	<i>null</i>

Interface Document

The Document interface allows access to XML documents as whole objects.

```
interface Document : Node {
    readonly attribute DocumentType      doctype;
    readonly attribute DOMImplementation implementation;
    readonly attribute Element           documentElement;
    Element                             createElement(in DOMString tagName)
                                         raises(DOMException);
    DocumentFragment                    createDocumentFragment();
    Text                                 createTextNode(in DOMString data);
    Comment                              createComment(in DOMString data);
    CDATASection                         createCDATASection(in DOMString data)
                                         raises(DOMException);
    ProcessingInstruction                createProcessingInstruction(in DOMString
target,
                                                                    in DOMString data)
                                         raises(DOMException);
    Attr                                 createAttribute(in DOMString name)
                                         raises(DOMException);
    EntityReference                      createEntityReference(in DOMString name)
                                         raises(DOMException);
    NodeList                             getElementsByTagName(in DOMString tagName);
};
```

Interface Element

Each XML element is mapped to an instance of an Element object. The definition is:

```
interface Element : Node {
    readonly attribute DOMString         tagName;
    DOMString                           getAttribute(in DOMString name);
    void                                 setAttribute(in DOMString name,
in DOMString value)
                                         raises(DOMException);
    void                                 removeAttribute(in DOMString name)
                                         raises(DOMException);
    Attr                                 getAttributeNode(in DOMString name);
    Attr                                 setAttributeNode(in Attr newAttr)
                                         raises(DOMException);
    Attr                                 removeAttributeNode(in Attr oldAttr)
                                         raises(DOMException);
    NodeList                             getElementsByTagName(in DOMString name);
    void                                 normalize();
};
```

Interface Attr

Individual XML attributes are accessed via the Attr interface.

```
interface Attr : Node {
    readonly attribute DOMString         name;
    readonly attribute boolean          specified;
```

```

        attribute DOMString          value;
};

```

Fields:

Type	Field Name	Explanation
DOMString	name	name of the attribute
boolean	specified	true if attribute is explicitly defined in the document
DOMString	value	value of the attribute

The `Attr` interface provides only the three fields that store information about a single attribute. Because attributes are accessed from `Element` objects via the `NamedNodeMap` interface, main access/creation methods to attributes in the `Element` and `NamedNodeMap` interfaces.

interface CharacterData

This is the base interface for character type nodes. The definition is:

```

interface CharacterData : Node {
    attribute DOMString          data;
    readonly attribute unsigned long length;

    DOMString substringData(in unsigned long offset,
                            in unsigned long count)
        raises(DOMException);
    void appendData(in DOMString arg)
        raises(DOMException);
    void insertData(in unsigned long offset,
                  in DOMString arg)
        raises(DOMException);
    void deleteData(in unsigned long offset,
                  in unsigned long count)
        raises(DOMException);
    void replaceData(in unsigned long offset,
                   in unsigned long count,
                   in DOMString arg)
        raises(DOMException);
};

```

This interface provides access to direct string manipulation methods.

Interface Comment. Interface `Comment` inherits all its properties from interface `CharacterData`.

Interface Text

Interface `Text` represents leaf nodes of the XML document (DOM tree), which contain only `#PCDATA` in them. This interface extends `CharacterData` with a single method `splitText()` for breaking off portions of the content of the text node into a new text node.

```

interface Text : CharacterData {
    Text splitText(in unsigned long offset)
};

```

Interface CDATASection. Interface `CDATASection` inherits *all* its properties from `Text`. This interface represents `CDATA` (unparsed character data)

sections in the XML document. (typical XML documents generated for data management tasks rarely contain CDATA sections).

```
interface CDATASection : Text {  
};
```

Interface DocumentType

This interface provides access to the actual DTD content.

```
interface DocumentType : Node {  
    readonly attribute DOMString      name;  
    readonly attribute NamedNodeMap   entities;  
    readonly attribute NamedNodeMap   notations;  
};
```

DOMString	name	name of the DTD
NamedNodeMap	entities	collection of all defined entities
NamedNodeMap	notations	collection of all defined notations

Interfaces NodeList and NamedNodeMap

Two more interfaces are defined *outside* the `Node` hierarchy, but closely connected with the `Node` interface.

`NodeList` interface defines the functionality for accessing an *ordered list* of `Node` objects. For each node, all its children are kept in order of their occurrence in the XML document, and thus, form a node list.

`NamedNodeMap` interface defines the functionality for accessing an *unordered collection* of `Node` objects. Since XML does not order attributes within a specific tag, a list of all attributes for a given XML element is a `NamedNodeMap` instance.

Interface NodeList

```
interface NodeList {  
    Node          item(in unsigned long index);  
    readonly attribute unsigned long length;  
};
```

`item(index)` returns the `Node` object which resides at position `index` in the list.

`length` is the total number of nodes in the list.

Interface NamedNodeMap

```
interface NamedNodeMap {  
    Node          getNamedItem(in DOMString name);  
    Node          setNamedItem(in Node arg)  
    Node          removeNamedItem(in DOMString name)  
    Node          item(in unsigned long index);  
  
    readonly attribute unsigned long length;  
};
```

`length` is the total number of elements in the collection.

`getNamedItem()` and `setNamedItem()` `removeNamedItem()` encapsulate access to the “named item” (i.e., a `Node` that has a name) by name.

`item(name)` retrieves the actual `Node` object by name.