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\footnote{Namespaces 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.} 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.
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 Type Structure Diagram]

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:

- interface Document: representation of the entire document
- interface Element: XML element nodes
- interface Attr: XML attribute nodes
- interface DocumentType: DTD nodes
- interface Comment: comments
- interface CharacterData: any parsed character data in the document
- interface Text: CharacterData: text nodes
- interface CDATASection: 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:

```javascript
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;

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

The brief explanation of the fields defined in this interface is below:
### Type Attribute Explanation

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

### Here are the methods:

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

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

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

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

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

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

```javascript
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:

```javascript
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 tagname);
  void normalize();
};
```

Interface Attr

Individual XML attributes are accessed via the Attr interface.

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

Fields:

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMString</td>
<td>name</td>
<td>name of the attribute</td>
</tr>
<tr>
<td>boolean</td>
<td>specified</td>
<td>true if attribute is explicitly defined in the document</td>
</tr>
<tr>
<td>DOMString</td>
<td>value</td>
<td>value of the attribute</td>
</tr>
</tbody>
</table>

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).

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

**Interface DocumentType**

This interface provides access to the actual DTD content.

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMString</td>
<td>name</td>
<td>name of the DTD</td>
</tr>
<tr>
<td>NamedNodeMap</td>
<td>entities</td>
<td>collection of all defined entities</td>
</tr>
<tr>
<td>NamedNodeMap</td>
<td>notations</td>
<td>collection of all defined notations</td>
</tr>
</tbody>
</table>

**Interfaces NodeList and NamedNodeMap**

Two more interfaces are defined outside the Node hierarchy, by 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**

```java
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**

```java
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.