Document Object Model (DOM) Level 3 Core Specification

Version 1.0

W3C Working Draft *01 September, 2000*

This version:
http://www.w3.org/TR/2000/WD-DOM-Level-3-Core-20000901
(PostScript file, PDF file, plain text, ZIP file)
Latest version:
http://www.w3.org/TR/DOM-Level-3-Core

Editors:
Arnaud Le Hors, *IBM*

Abstract

This specification defines the Document Object Model Core Level 3, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model Core Level 3 builds on the Document Object Model Core Level 2.

Status of this document

This document is a preliminary version of the Level 3 API.

It is a W3C Working Draft for review by W3C members and other interested parties and acts as a starting point for the future DOM Working Group, should it be approved or not by the W3C Members. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress".

Comments on this document are invited and are to be sent to the public mailing list www-dom@w3.org. An archive is available at http://lists.w3.org/Archives/Public/www-dom/.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Table of contents

Expanded Table of Contents

Expanded Table of Contents

Copyright Notice

This document is published under the W3C Document Copyright Notice and License [p.5] . The bindings within this document are published under the W3C Software Copyright Notice and License [p.6] . The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL binding, the pragma prefix can no longer be 'w3c.org'; in the case of the Java binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-documents-19990405.

http://www.w3.org/Consortium/Legal/

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

1. Document Object Model Core

Editors

Arnaud Le Hors, IBM

1.1. DOM Level 3 Core

(***ED:*** Although the following defines a set of new interfaces that extend the DOM Level 2 interfaces the current plan is not to actually add any new interfaces but instead to expand the existing ones.)

Issue Level-3-Core-1:

Instead of extending the core interfaces should we define some kind of DOMUtility interface which could be optional?

Interface *Entity3*

This interface extends the `Entity` interface with additional attributes to provide information on the text declaration of external parsed entities.

IDL Definition

```
interface Entity3 : Entity {
           attribute DOMString        actualEncoding;
           attribute DOMString        encoding;
           attribute DOMString        version;
};
```

Attributes

`actualEncoding` of type `DOMString`

An attribute specifying the actual encoding of this entity, when it is an external parsed entity. This is `null` otherwise.

`encoding` of type `DOMString`

An attribute specifying, as part of the text declaration, the encoding of this entity, when it is an external parsed entity. This is `null` otherwise.

`version` of type `DOMString`

An attribute specifying, as part of the text declaration, the version number of this entity, when it is an external parsed entity. This is `null` otherwise.

Interface *Document3*

This interface extends the `Document` interface with additional attributes and methods.

IDL Definition

```
interface Document3 : Document {
           attribute DOMString        actualEncoding;
           attribute DOMString        encoding;
           attribute boolean          standalone;
           attribute boolean          strictErrorChecking;
           attribute DOMString        version;
```

```
  Node              adoptNode(in Node source)
                                        raises(DOMException);
  NodeList          getElementsByAttributeValue(in DOMString namespaceURI,
                                                in DOMString localName,
                                                in DOMString value);
};
```

Attributes

`actualEncoding` of type `DOMString`
: An attribute specifying the actual encoding of this document. This is `null` otherwise.

`encoding` of type `DOMString`
: An attribute specifying, as part of the XML declaration, the encoding of this document. This is `null` when unspecified.

`standalone` of type `boolean`
: An attribute specifying, as part of the XML declaration, whether this document is standalone.

`strictErrorChecking` of type `boolean`
: An attribute specifying whether errors checking is enforced or not. When set to `false`, the implementation is free to not test every possible error case normally defined on DOM operations, and not raise any `DOMException`. In case of error, the behavior is undefined. This attribute is `true` by defaults.

`version` of type `DOMString`
: An attribute specifying, as part of the XML declaration, the version number of this document. This is `null` when unspecified.

Methods

`adoptNode`
: Changes the `ownerDocument` of a node, its children, as well as the attached attribute nodes if there are any. If the node has a parent it is first removed from its parent child list. This effectively allows moving a subtree from one document to another. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE
: The `ownerElement` attribute is set to `null` and the `specified` flag is set to `true` on the adopted `Attr`. The descendants of the source `Attr` are recursively adopted.

DOCUMENT_FRAGMENT_NODE
: The descendants of the source node are recursively adopted.

DOCUMENT_NODE
: `Document` nodes cannot be adopted.

DOCUMENT_TYPE_NODE
: `DocumentType` nodes cannot be adopted.

ELEMENT_NODE
: *Specified* attribute nodes of the source element are adopted, and the generated `Attr` nodes. Default attributes are discarded, though if the document being adopted into defines default attributes for this element name, those are assigned. The descendants

of the source element are recursively adopted.

ENTITY_NODE

`Entity` nodes can be adopted, however in the current release of the DOM the `DocumentType` is readonly. Ability to add these adopted nodes to a `DocumentType` will be considered for addition to a future release of the DOM. The descendants of the the source entity are recursively adopted.

ENTITY_REFERENCE_NODE

Only the `EntityReference` node itself is adopted, the descendants are discarded, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE

`Notation` nodes can be adopted, however in the current release of the DOM the `DocumentType` is readonly. Ability to add these adopted nodes to a `DocumentType` will be considered for addition to a future release of the DOM.

PROCESSING_INSTRUCTION_NODE, TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These nodes can all be adopted. No specifics.

Issue adoptNode-1:

Should this method simply return null when it fails? How "exceptional" is failure for this method?

Resolution: Stick with raising exceptions only in exceptional circumstances, return null on failure (F2F 19 Jun 2000).

Issue adoptNode-2:

Can an entity node really be adopted?

Parameters

`source` of type `Node`

The node to move into this document.

Return Value

`Node`	The adopted node, or `null` if this operation fails, such as when the source node comes from a different implementation.

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if the source node is of type `DOCUMENT`, `DOCUMENT_TYPE`. NO_MODIFICATION_ALLOWED_ERR: Raised when the source node is readonly.

`getElementsByAttributeValue`

Get a list of all the elements which have a specific attribute.

Issue getElementsByAttributeValue-1:
Anyone still lobbying for this?
Resolution: No, killed! (F2F 1 Aug 2000)
Issue getElementsByAttributeValue-2:
Should we have getElementsByAttributeValue and getElementsByAttributeValueNS?
Resolution: Yes, but irrelevant, see above. (F2F 1 Aug 2000)
Issue getElementsByAttributeValue-3:
Should we have this on Element as well? We have getElementsByTagName there.
Resolution: Irrelevant, see above. (F2F 1 Aug 2000)

Parameters

`namespaceURI` of type `DOMString`
The namespace URI of the attribute to look for.

`localName` of type `DOMString`
The local name of the attribute to look for.

`value` of type `DOMString`
The value of the attribute to look for.

Return Value

`NodeList`	Returns a list containing the elements that have an attribute that matches the given namespace URI, local name, and value.

No Exceptions

Interface *Node3*

This interface extends the `Node` interface with several new methods. One allows to compare a node against another with regard to document order. Another methode allows to retrieve the content of a node and its descendants as a single DOMString. One allows to test whether two nodes are the same. Two methods provide for searching the namespace URI associated to a given prefix and to ensure the document is "namespace wellformed". Finally, a method allows to attach some user data to a Node. It also provides a new attribute to get the base URI of a node, as defined in the XML Infoset.
Issue namespace-wellformed:
The term *namespace wellformed* needs to be defined.
Resolution: Define it as "being conformant to the Namespaces in XML spec" in the glossary (Telcon 4 Jul 2000).

IDL Definition

```
interface Node3 {
  readonly attribute DOMString        baseURI;

  typedef enum _DocumentOrder {
    PRECEDING,
    FOLLOWING,
    SAME,
    UNORDERED
  };
```

```
 DocumentOrder;
  DocumentOrder      compareDocumentOrder(in Node other)
                                        raises(DOMException);

  typedef enum _TreePosition {
    PRECEDING,
    FOLLOWING,
    ANCESTOR,
    DESCENDANT,
    SAME,
    UNORDERED
  };
 TreePosition;
  TreePosition       compareTreePosition(in Node other)
                                        raises(DOMException);
           attribute DOMString        textContent;
  boolean            isSameNode(in Node other);
  DOMString          lookupNamespacePrefix(in DOMString namespaceURI);
  DOMString          lookupNamespaceURI(in DOMString prefix);
  void               normalizeNS();
  Object             setUserData(in Object data,
                                 in DOMString key);
  Object             getUserData(in DOMString key);
  readonly attribute Object           key;
};
```

Type Definition *DocumentOrder*

A type to hold the document order of a node relative to another node.
Enumeration *_DocumentOrder*

An enumeration of the different orders the node can be in.

Enumerator Values

PRECEDING	The node preceds the reference node in document order.
FOLLOWING	The node follows the reference node in document order.
SAME	The two nodes have the same document order.
UNORDERED	The two nodes are unordered, they do not have any common ancestor.

Type Definition *TreePosition*

A type to hold the relative tree position of a node with respect to another node.
Enumeration *_TreePosition*

An enumeration of the different orders the node can be in.

Enumerator Values

PRECEDING	The node preceds the reference node.
FOLLOWING	The node follows the reference node.
ANCESTOR	The node is an ancestor of the reference node.
DESCENDANT	The node is a descendant of the reference node.
SAME	The two nodes have the same position.
UNORDERED	The two nodes are unordered, they do not have any common ancestor.

Attributes

`baseURI` of type `DOMString`, readonly

Returns the absolute base URI of this node.

Issue baseURI-1:

How will this be affected by resolution of relative namespace URIs issue?

Issue baseURI-2:

Should this only be on Document, Element, ProcessingInstruction, Entity, and Notation nodes, according to the infoset? If not, what is it equal to on other nodes? Null? An empty string?

Issue baseURI-3:

Should this be read-only and computed or and actual read-write attribute?
Resolution: The former (F2F 19 Jun 2000).

`key` of type `Object`, readonly

This attribute returns a unique key identifying this node.

Issue key-1:

What type should this really be?

Issue key-2:

In what space is this key unique (Document, DOMImplementation)?

Issue key-3:

What is the lifetime of the uniqueness of this key (Node, Document, ...)?

`textContent` of type `DOMString`

This attribute returns the text content of this node and its descendants. When set, any possible children this node may have are removed and replaced by a single `Text` node containing the string this attribute is set to. On getting, no serialization is performed, the returned string does not contain any markup. Similarly, on setting, no parsing is performed either, the input string is taken as pure textual content.
The string returned is made of the text content of this node depending on its type, as

defined below:

Node type	Content
ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, DOCUMENT_NODE, DOCUMENT_FRAGMENT_NODE	concatenation of the `textContent` of the child nodes, excluding COMMENT_NODE and PROCESSING_INSTRUCTION_NODE nodes
ATTRIBUTE_NODE, TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE, PROCESSING_INSTRUCTION_NODE	`nodeValue`
DOCUMENT_TYPE_NODE, NOTATION_NODE	*empty string*

Issue textContent-1:
: Should any whitespace normalization be performed?

Issue textContent-2:
: Should this be two methods instead?

Issue textContent-3:
: What about the name?

Methods

`compareDocumentOrder`

Compares a node with this node with regard to document order.

Issue compareOrder-1:
: Should an exception be raised when comparing attributes? Entities and notations? An element against an attribute? If yes, which one? HIERARCHY_REQUEST_ERR? Should the enum value "unordered" be killed then?
Resolution: No, return unordered for attributes (F2F 19 Jun 2000).

Issue compareOrder-2:
: Should this method be moved to Node and take only one node in argument?
Resolution: Yes (F2F 19 Jun 2000).

Parameters

`other` of type `Node`
: The node to compare against this node.

Return Value

`DocumentOrder` [p.13]	Returns how the given node compares with this node in document order.

Exceptions

`DOMException`	WRONG_DOCUMENT_ERR: Raised if the given node does not belong to the same document as this node.

`compareTreePosition`

Compares a node with this node with regard to their position in the tree.

Parameters

`other` of type `Node`

The node to compare against this node.

Return Value

`TreePosition` [p.13]	Returns how the given node is positioned relatively to this node.

Exceptions

`DOMException`	WRONG_DOCUMENT_ERR: Raised if the given node does not belong to the same document as this node.

`getUserData`

This method allows to retreive a user object previously attached to a Node with `setUserData`.

Parameters

`key` of type `DOMString`

The key to look for.

Return Value

`Object`	The object associated to this node and the given key or `null`.

No Exceptions

`isSameNode`

Returns whether this node is the same node as the given one.

Issue isIdentical-1:

Do we really want to make this different from equals?

Resolution: Yes, change name from isIdentical to isSameNode. (Telcon 4 Jul 2000).

Parameters

`other` of type `Node`

The node to test against.

Return Value

`boolean`	Returns `true` if the nodes are the same, `false` otherwise.

No Exceptions

`lookupNamespacePrefix`

Look up the prefix associated to the given namespace URI, starting from this node.

Parameters

`namespaceURI` of type `DOMString`

The namespace URI to look for.

Return Value

`DOMString`	Returns the associated namespace prefix or `null` if none is found.

No Exceptions

`lookupNamespaceURI`

Look up the namespace URI associated to the given prefix, starting from this node.

Issue lookupNamespaceURI-1:

Name? May need to change depending on ending of the relative namespace URI reference nightmare.

Parameters

`prefix` of type `DOMString`

The prefix to look for.

Return Value

`DOMString`	Returns the associated namespace URI or `null` if none is found.

No Exceptions

`normalizeNS`

This method walks down the tree, starting from this node, and adds namespace declarations where needed so that every namespace being used is properly declared. It also changes or assign prefixes when needed. This effectively makes this node subtree is "namespace wellformed".

What the generated prefixes are and/or how prefixes are changed to achieve this is implementation dependent.

Issue normalizeNS-1:

Any other name?

Issue normalizeNS-2:

How specific should this be? Should we not even specify that this should be done by walking down the tree?

Issue normalizeNS-3:
: What does this do on attribute nodes?
: **Resolution:** Doesn't do anything (F2F 1 Aug 2000).

Issue normalizeNS-4:
: How does it work with entity reference subtree which may be broken?
: **Resolution:** This doesn't affect entity references which are not visited in this operation (F2F 1 Aug 2000).

Issue normalizeNS-5:
: Should this be really be on Node?
: **Resolution:** Yes, but this only works on Document, Element, and DocumentFragment. On other types it is a no-op. (F2F 1 Aug 2000).

Issue normalizeNS-6:
: What happens with read-only nodes?

Issue normalizeNS-7:
: What/how errors should be reported? Are there any?

No Parameters
No Return Value
No Exceptions

`setUserData`

This method allows to attach a user object to a Node. This object can then be retreived using `getUserData`.

Issue setUserData-1:
: Is this really worth it?

Issue setUserData-2:
: Could we live without a key?

Issue setUserData-3:
: What happens if the node is cloned, imported, adopted? Should some event mechanism be specified to notify the application?

Issue setUserData-4:
: What happens if the node is cloned?

Issue setUserData-5:
: What should Object be mapped to in ECMAScript? For IDL we need to define this type somehow.

Parameters

`data` of type `Object`
: The piece of data to attach to this node.

`key` of type `DOMString`
: The key to associate this data to.

Return Value

`Object`	The object previously associated to this node and the given key or `null`.

No Exceptions

Interface *Text3*

This interface extends the `Text` interface with a new attribute that allows one to find out whether a `Text` node only contains whitespace in element content.

IDL Definition

```
interface Text3 : Text {
  readonly attribute boolean          isWhitespaceInElementContent;
};
```

Attributes

`isWhitespaceInElementContent` of type `boolean`, readonly
: Returns whether this text node contains whitespace in element content, often abusively called "ignorable whitespace".

  Note: An implementation can only return `true` if, one way or another, it has access to the relevant information (e.g., the DTD or schema).

Appendix A: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 3 Document Object Model Core definitions.

The IDL files are also available as:
http://www.w3.org/TR/2000/WD-DOM-Level-3-Core-20000901/idl.zip

dom.idl:

```
// File: dom.idl

#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix "w3c.org"
module dom
{

  typedef null::DOMString DOMString;
  typedef null::Object Object;
  typedef null::Node Node;
  typedef null::Entity Entity;
  typedef null::Document Document;
  typedef null::NodeList NodeList;
  typedef null::Text Text;

  interface Node3 {
    readonly attribute DOMString        baseURI;

    typedef enum _DocumentOrder {
      PRECEDING,
      FOLLOWING,
      SAME,
      UNORDERED
    };
 DocumentOrder;
    DocumentOrder      compareDocumentOrder(in Node other)
                                        raises(DOMException);

    typedef enum _TreePosition {
      PRECEDING,
      FOLLOWING,
      ANCESTOR,
      DESCENDANT,
      SAME,
      UNORDERED
    };
 TreePosition;
    TreePosition       compareTreePosition(in Node other)
                                        raises(DOMException);
             attribute DOMString        textContent;
    boolean            isSameNode(in Node other);
    DOMString          lookupNamespacePrefix(in DOMString namespaceURI);
```

```
    DOMString          lookupNamespaceURI(in DOMString prefix);
    void               normalizeNS();
    Object             setUserData(in Object data,
                                   in DOMString key);
    Object             getUserData(in DOMString key);
    readonly attribute Object           key;
  };

  interface Entity3 : Entity {
             attribute DOMString        actualEncoding;
             attribute DOMString        encoding;
             attribute DOMString        version;
  };

  interface Document3 : Document {
             attribute DOMString        actualEncoding;
             attribute DOMString        encoding;
             attribute boolean          standalone;
             attribute boolean          strictErrorChecking;
             attribute DOMString        version;
    Node               adoptNode(in Node source)
                                        raises(DOMException);
    NodeList           getElementsByAttributeValue(in DOMString namespaceURI,
                                                   in DOMString localName,
                                                   in DOMString value);
  };

  interface Text3 : Text {
    readonly attribute boolean          isWhitespaceInElementContent;
  };
};

#endif // _DOM_IDL_
```

Appendix B: Java Language Binding

This appendix contains the complete Java [Java] bindings for the Level 3 Document Object Model Core.

The Java files are also available as
http://www.w3.org/TR/2000/WD-DOM-Level-3-Core-20000901/java-binding.zip

org/w3c/dom/Entity3.java:

```
package org.w3c.dom;

import org.w3c.Entity;

public interface Entity3 extends Entity {
    public String getActualEncoding();
    public void setActualEncoding(String actualEncoding);

    public String getEncoding();
    public void setEncoding(String encoding);

    public String getVersion();
    public void setVersion(String version);

}
```

org/w3c/dom/Document3.java:

```
package org.w3c.dom;

import org.w3c.Node;
import org.w3c.Document;
import org.w3c.DOMException;
import org.w3c.NodeList;

public interface Document3 extends Document {
    public String getActualEncoding();
    public void setActualEncoding(String actualEncoding);

    public String getEncoding();
    public void setEncoding(String encoding);

    public boolean getStandalone();
    public void setStandalone(boolean standalone);

    public boolean getStrictErrorChecking();
    public void setStrictErrorChecking(boolean strictErrorChecking);

    public String getVersion();
    public void setVersion(String version);

    public Node adoptNode(Node source)
                          throws DOMException;

    public NodeList getElementsByAttributeValue(String namespaceURI,
```

```
                                                     String localName,
                                                     String value);

}
```

org/w3c/dom/Node3.java:

```
package org.w3c.dom;

import org.w3c.Object;
import org.w3c.Node;
import org.w3c.DocumentOrder;
import org.w3c.TreePosition;
import org.w3c.DOMException;

public interface Node3 {
    public String getBaseURI();


    public static final int PRECEDING    = 1;
    public static final int FOLLOWING    = 2;
    public static final int SAME         = 3;
    public static final int UNORDERED    = 4;

    public int compareDocumentOrder(Node other)
                                    throws DOMException;


    public static final int PRECEDING    = 1;
    public static final int FOLLOWING    = 2;
    public static final int ANCESTOR     = 3;
    public static final int DESCENDANT   = 4;
    public static final int SAME         = 5;
    public static final int UNORDERED    = 6;

    public int compareTreePosition(Node other)
                                   throws DOMException;

    public String getTextContent();
    public void setTextContent(String textContent);

    public boolean isSameNode(Node other);

    public String lookupNamespacePrefix(String namespaceURI);

    public String lookupNamespaceURI(String prefix);

    public void normalizeNS();

    public Object setUserData(Object data,
                              String key);

    public Object getUserData(String key);
```

```
    public Object getKey();

}
```

org/w3c/dom/Text3.java:

```
package org.w3c.dom;

import org.w3c.Text;

public interface Text3 extends Text {
    public boolean getIsWhitespaceInElementContent();

}
```

org/w3c/dom/Text3.java:

Appendix C: ECMA Script Language Binding

This appendix contains the complete ECMA Script [ECMAScript] binding for the Level 3 Document Object Model Core definitions.

Object **Entity3**

Entity3 has the all the properties and methods of **Entity** as well as the properties and methods defined below.

The **Entity3** object has the following properties:

actualEncoding

This property is of type **String**.

encoding

This property is of type **String**.

version

This property is of type **String**.

Object **Document3**

Document3 has the all the properties and methods of **Document** as well as the properties and methods defined below.

The **Document3** object has the following properties:

actualEncoding

This property is of type **String**.

encoding

This property is of type **String**.

standalone

This property is of type **boolean**.

strictErrorChecking

This property is of type **boolean**.

version

This property is of type **String**.

The **Document3** object has the following methods:

adoptNode(source)

This method returns a **Node**.

The **source** parameter is of type **Node**.

getElementsByAttributeValue(namespaceURI, localName, value)

This method returns a **NodeList**.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

The **value** parameter is of type **String**.

Object **Node3**

The **Node3** object has the following properties:

baseURI

This property is of type **String**.

textContent

This property is of type **String**.

key
This property is of type **Object**.

The **Node3** object has the following methods:

compareDocumentOrder(other)
This method returns a **DocumentOrder**.
The **other** parameter is of type **Node**.

compareTreePosition(other)
This method returns a **TreePosition**.
The **other** parameter is of type **Node**.

isSameNode(other)
This method returns a **boolean**.
The **other** parameter is of type **Node**.

lookupNamespacePrefix(namespaceURI)
This method returns a **String**.
The **namespaceURI** parameter is of type **String**.

lookupNamespaceURI(prefix)
This method returns a **String**.
The **prefix** parameter is of type **String**.

normalizeNS()
This method has no return value.

setUserData(data, key)
This method returns a **Object**.
The **data** parameter is of type **Object**.
The **key** parameter is of type **String**.

getUserData(key)
This method returns a **Object**.
The **key** parameter is of type **String**.

Object **Text3**

Text3 has the all the properties and methods of **Text** as well as the properties and methods defined below.

The **Text3** object has the following properties:

isWhitespaceInElementContent
This property is of type **boolean**.

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at http://www.w3.org/TR.

D.1: Normative references

ECMAScript
ECMA (European Computer Manufacturers Association) ECMAScript Language Specification. Available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

Java
Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at http://java.sun.com/docs/books/jls

OMGIDL
OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available at http://sisyphus.omg.org/technology/documents/formal/corba_2.htm

D.1: Normative references

Index