Document Object Model (DOM) Level 3 Validation Specification

Version 1.0

W3C Recommendation 27 January 2004

This version:
http://www.w3.org/TR/2004/REC-DOM-Level-3-Val-20040127
Latest version:
http://www.w3.org/TR/DOM-Level-3-Val
Previous version:
http://www.w3.org/TR/2003/PR-DOM-Level-3-Val-20031215

Editors:
Ben Chang, *Oracle*
Joe Kesselman, *IBM (until September 2001)*
Rezaur Rahman, *Intel Corporation (until July 2001)*

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: XML file, plain text, PostScript file, PDF file, single HTML file, and ZIP file.

See also translations of this document.

Abstract

This specification defines the Document Object Model Validation Level 3, a platform- and language-neutral interface. This module provides the guidance to programs and scripts to dynamically update the content and the structure of documents while ensuring that the document remains valid, or to ensure that the document becomes valid.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is the Recommendation of "DOM Level 3 Validation". No changes were done to this specification since the Proposed Recommendation version.

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

An implementation report, along with the test suite, is also available.

This is a stable document and has been endorsed by the W3C Membership and the participants of the DOM working group. The English version of this specification is the only normative version. Information about translations of this document is available at http://www.w3.org/2004/01/DOM-Level-3-translations.

The list of known errors in this document is available at http://www.w3.org/2004/01/DOM-Level-3-errata.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM Working Group participants.

Patent disclosures relevant to this specification may be found on the Working Group’s patent disclosure page.

Table of contents

Expanded Table of Contents

W3C Copyright Notices and Licenses

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 definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java language 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/2002/copyright-documents-20021231.

http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Public documents on the W3C site are provided by the copyright holders under the following license. 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 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 (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
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.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those

requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

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/2002/copyright-software-20021231

Copyright © 2004 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C® Short Software Notice [p.7] should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

W3C® Short Software Notice

Note: This section is a copy of the W3C® Short Software Notice and could be found at http://www.w3.org/Consortium/Legal/2002/copyright-software-short-notice-20021231

Copyright © 2004 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

Copyright © [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. This work is distributed under the W3C® Software License [1] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

1. Validation

Editors:
Ben Chang, Oracle
Joe Kesselman, IBM (until September 2001)
Rezaur Rahman, Intel Corporation (until July 2001)

1.1 Overview

This chapter describes the optional DOM Level 3 Validation feature. This module provides Application Programming Interfaces (APIs) to guide construction and editing of XML documents. Examples of such guided editing are queries like those that combine questions like "what does the schema allow me to insert/delete here" and "if I insert/delete here, will the document still be valid."

To aid users in the editing and creation of XML documents, other queries may expose different levels of details, e.g., all the possible children, those which would be valid given what precedes this point, lists of defined symbols of a given kind. Some of these queries would prompt checks and warn users if they’re about to conflict with or overwrite such data.

Finally, users would like to validate an edited or newly constructed document before serializing it or passing it to other users. They may edit, come up with an invalid document, then edit again to result in a valid document. During this process, these APIs can allow the user to check the validity of the document or subtree on demand. If necessary, these APIs can also require that the document or subtree remain valid during this editing process via the `DocumentEditVal.continuousValidityChecking` flag.

A DOM application can use the `hasFeature(feature, version)` method of the `DOMImplementation` interface to determine with parameter values `"Validation"` and `"3.0"`, respectively, whether or not these interfaces are supported by the implementation. This implementation is dependent on [DOM Level 2 Core] and the [DOM Level 3 Core] `DOMConfiguration` interface.

This chapter focuses on the editing aspects used in the XML document editing world and usage of such information. The appendix describes in detail all the possible outcomes of the validation operations on the different node types.

1.2 Exceptions

This section describes the "VAL-DOC-EDIT" exceptions.

Exception *ExceptionVAL*

Some Validation operations may throw an `ExceptionVAL` [p.9] as described in their descriptions.
IDL Definition

```
exception ExceptionVAL {
  unsigned short   code;
};
// ExceptionVALCode
const unsigned short      NO_SCHEMA_AVAILABLE_ERR        = 71;
```

Definition group *ExceptionVALCode*

An integer indicating the type of error generated.

Defined Constants

`NO_SCHEMA_AVAILABLE_ERR`

This error occurs when the operation cannot complete due to an unavailable schema [p.41] .

1.3 Document Editing Interfaces

This section contains "Document Editing" methods as described in the `DocumentEditVAL` [p.10] , `NodeEditVAL` [p.11] , `ElementEditVAL` [p.14] , and `CharacterDataEditVAL` [p.19] interfaces. References to new [DOM Level 3 Core] interfaces such as `DOMStringList` and `NameList` also exist. With the latter interface, if the schema is a DTD, the element information item names are simply local names; if the schema is a W3C XML schema, the names are qualified names, which may contain namespace prefixes.

Interface *DocumentEditVAL*

This interface extends the `NodeEditVAL` [p.11] interface with additional methods for document editing. An object implementing this interface must also implement the `Document` interface.

IDL Definition

```
interface DocumentEditVAL : NodeEditVAL {
           attribute boolean         continuousValidityChecking;
                                        // raises(DOMException,
                                        //        ExceptionVAL,
                                        //        DOMException) on setting

  readonly attribute DOMConfiguration domConfig;
  NameList           getDefinedElements(in DOMString namespaceURI);
  unsigned short     validateDocument();
};
```

Attributes

`continuousValidityChecking` of type `boolean`

An attribute specifying whether the validity of the document is continuously enforced. When the attribute is set to `true`, the implementation may raise certain exceptions, depending on the situation (see the following). This attribute is `false` by default.

Exceptions on setting

`DOMException`	NOT_SUPPORTED_ERR: Raised if the implementation does not support setting this attribute to `true`.
`ExceptionVAL` [p.9]	NO_SCHEMA_AVAILABLE_ERR: Raised if this attribute is set to `true` and a schema is unavailable.
`DOMException`	VALIDATION_ERR: Raised if an operation makes this document not compliant with the `VAL_INCOMPLETE` validity type or the document is invalid, and this attribute is set to `true`.

`domConfig` of type `DOMConfiguration`, readonly

This allows the setting of the error handler, as described in the [DOM Level 3 Core] `DOMConfiguration` interface. An object implementing this `DocumentEditVAL` interface and the [DOM Level 3 Core] `Document` interface, which also has a `domConfig` attribute, needs to only implement this attribute once.

Methods

`getDefinedElements`

Returns list of all element information item names of global declaration [p.41] , belonging to the specified namespace.

Parameters

`namespaceURI` of type `DOMString`

`namespaceURI` of namespace. For DTDs, this is `null`.

Return Value

`NameList`	List of all element information item names belonging to the specified namespace or `null` if no schema is available.

No Exceptions

`validateDocument`

Validates the document against the schema, e.g., a DTD or an W3C XML schema or another. Any attempt to modify any part of the document while validating results in implementation-dependent behavior. In addition, the validation operation itself cannot modify the document, e.g., for default attributes. This method makes use of the error handler, as described in the [DOM Level 3 Core] `DOMConfiguration` interface, with all errors being `SEVERITY_ERROR` as defined in the `DOMError` interface.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Parameters

No Exceptions

Interface *NodeEditVAL*

This interface is similar to the [DOM Level 3 Core] `Node` interface, with methods for guided document editing.

IDL Definition

```
interface NodeEditVAL {

  // validationType
  const unsigned short      VAL_WF                        = 1;
  const unsigned short      VAL_NS_WF                     = 2;
  const unsigned short      VAL_INCOMPLETE                = 3;
  const unsigned short      VAL_SCHEMA                    = 4;


  // validationState
  const unsigned short      VAL_TRUE                      = 5;
  const unsigned short      VAL_FALSE                     = 6;
  const unsigned short      VAL_UNKNOWN                   = 7;

  readonly attribute DOMString       defaultValue;
  readonly attribute DOMStringList   enumeratedValues;
  unsigned short     canInsertBefore(in Node newChild,
                                     in Node refChild);
  unsigned short     canRemoveChild(in Node oldChild);
  unsigned short     canReplaceChild(in Node newChild,
                                     in Node oldChild);
  unsigned short     canAppendChild(in Node newChild);
  unsigned short     nodeValidity(in unsigned short valType);
};
```

Definition group *validationType*

An integer indicating the validation type. Other specifications can define stricter validation types/constants by extending the `NodeEditVAL` interface.

Defined Constants

`VAL_INCOMPLETE`
: Check if the node's immediate children are those expected by the content model. This node's trailing required children could be missing. It includes `VAL_NS_WF`.

`VAL_NS_WF`
: Check if the node is namespace well-formed [p.41] .

`VAL_SCHEMA`
: Check if the node's entire subtree are those expected by the content model. It includes `VAL_NS_WF`.

`VAL_WF`
: Check if the node is well-formed [p.41] .

Definition group *validationState*

An integer indicating the validation state, or whether the operation can or cannot be done.

Defined Constants

`VAL_FALSE`
: False if the node is invalid with regards to the operation, or if the operation cannot be done.

`VAL_TRUE`
: True if the node is valid with regards to the operation, or if the operation can be done.

`VAL_UNKNOWN`
: The validity of the node is unknown.

Attributes

`defaultValue` of type `DOMString`, readonly
: The default value specified in an attribute or an element declaration or `null` if unspecified. If the schema is a W3C XML schema, this is the canonical lexical representation of the default value.

`enumeratedValues` of type `DOMStringList`, readonly
: A `DOMStringList`, as described in [DOM Level 3 Core], of distinct values for an attribute or an element declaration or `null` if unspecified. If the schema is a W3C XML schema, this is a list of strings which are lexical representations corresponding to the values in the [value] property of the enumeration component for the type of the attribute or element. It is recommended that the canonical lexical representations of the values be used.

Methods

`canAppendChild`
: Determines whether the `Node.appendChild` operation would make this document not compliant with the `VAL_INCOMPLETE` validity type.

Parameters

`newChild` of type `Node`
: `Node` to be appended.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canInsertBefore`
: Determines whether the `Node.insertBefore` operation would make this document not compliant with the `VAL_INCOMPLETE` validity type.

Parameters

`newChild` of type `Node`
: `Node` to be inserted.

`refChild` of type `Node`
: Reference `Node`.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canRemoveChild`
: Determines whether the `Node.removeChild` operation would make this document not compliant with the `VAL_INCOMPLETE` validity type.

Parameters

`oldChild` of type `Node`
`Node` to be removed.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`canReplaceChild`
Determines whether the `Node.replaceChild` operation would make this document not compliant with the `VAL_INCOMPLETE` validity type.

Parameters

`newChild` of type `Node`
New `Node`.

`oldChild` of type `Node`
`Node` to be replaced.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`nodeValidity`
Determines if the node is valid relative to the validation type [p.12] specified in `valType`. This operation doesn't normalize before checking if it is valid. To do so, one would need to explicitly call a normalize method. The difference between this method and the DocumentEditVAL.validateDocument [p.11] method is that the latter method only checks to determine whether the entire document is valid.

Parameters

`valType` of type `unsigned short`
Flag to indicate the validation type [p.12] checking to be done.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

Interface *ElementEditVAL*

This interface extends the `NodeEditVAL` [p.11] interface with additional methods for guided document editing. An object implementing this interface must also implement the `Element` interface.

This interface also has attributes that are a `NameList` of elements or attributes which can appear in the specified context. Some schema languages, i.e., W3C XML schema, define wildcards which provide for validation of attribute and element information items dependent on their namespace names but independent of their local names.

To expose wildcards, the `NameList` returns the values that represent the namespace constraint:

- `{namespaceURI, name}` is `{null, ##any}` if *any*;
- `{namespaceURI, name}` is `{namespace_a, ##other}` if *not and a namespace name (namespace_a)*;
- `{namespaceURI, name}` is `{null, ##other}` if *not and absent*;
- Pairs of `{namespaceURI, name}` with values `{a_namespaceURI | null, null}` if *a set whose members are either namespace names or absent*.

IDL Definition

```
interface ElementEditVAL : NodeEditVAL {

  // ContentTypeVAL
  const unsigned short      VAL_EMPTY_CONTENTTYPE          = 1;
  const unsigned short      VAL_ANY_CONTENTTYPE            = 2;
  const unsigned short      VAL_MIXED_CONTENTTYPE          = 3;
  const unsigned short      VAL_ELEMENTS_CONTENTTYPE       = 4;
  const unsigned short      VAL_SIMPLE_CONTENTTYPE         = 5;

  readonly attribute NameList        allowedChildren;
  readonly attribute NameList        allowedFirstChildren;
  readonly attribute NameList        allowedParents;
  readonly attribute NameList        allowedNextSiblings;
  readonly attribute NameList        allowedPreviousSiblings;
  readonly attribute NameList        allowedAttributes;
  readonly attribute NameList        requiredAttributes;
  readonly attribute unsigned short  contentType;
  unsigned short     canSetTextContent(in DOMString possibleTextContent);
  unsigned short     canSetAttribute(in DOMString attrname,
                                     in DOMString attrval);
  unsigned short     canSetAttributeNode(in Attr attrNode);
  unsigned short     canSetAttributeNS(in DOMString namespaceURI,
                                       in DOMString qualifiedName,
                                       in DOMString value);
  unsigned short     canRemoveAttribute(in DOMString attrname);
  unsigned short     canRemoveAttributeNS(in DOMString namespaceURI,
                                          in DOMString localName);
  unsigned short     canRemoveAttributeNode(in Node attrNode);
  unsigned short     isElementDefined(in DOMString name);
  unsigned short     isElementDefinedNS(in DOMString namespaceURI,
                                        in DOMString name);
};
```

Definition group *ContentTypeVAL*

An integer indicating the content type of an element.

Defined Constants

`VAL_ANY_CONTENTTYPE`

The content model contains unordered child information item(s), i.e., element, processing instruction, unexpanded entity reference, character, and comment information items as defined in the XML Information Set. If the schema is a DTD, this corresponds to the `ANY` content model.

`VAL_ELEMENTS_CONTENTTYPE`
: The content model contains a sequence of element information items optionally separated by whitespace. If the schema is a DTD, this is the `element content` content model; and if the schema is a W3C XML schema, this is the `element-only` content type.

`VAL_EMPTY_CONTENTTYPE`
: The content model does not allow any content. If the schema is a W3C XML schema, this corresponds to the `empty` content type; and if the schema is a DTD, this corresponds to the `EMPTY` content model.

`VAL_MIXED_CONTENTTYPE`
: The content model contains a sequence of ordered element information items optionally interspersed with character data. If the schema is a W3C XML schema, this corresponds to the `mixed` content type.

`VAL_SIMPLE_CONTENTTYPE`
: The content model contains character information items. If the schema is a W3C XML schema, then the element has a content type of `VAL_SIMPLE_CONTENTTYPE` if the type of the element is a `simple type definition`, or the type of the element is a `complexType` whose `{content type}` is a `simple type definition`.

Attributes

`allowedAttributes` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all possible attribute information items or wildcards [p.14] that can appear as attributes of this element, or `null` if this element has no context or schema. Duplicate pairs of {namespaceURI, name} are eliminated.

`allowedChildren` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all possible element information items or wildcards [p.14] that can appear as children of this element, or `null` if this element has no context or schema. Duplicate pairs of {namespaceURI, name} are eliminated.

`allowedFirstChildren` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all possible element information items or wildcards [p.14] that can appear as a first child of this element, or `null` if this element has no context or schema. Duplicate pairs of {namespaceURI, name} are eliminated.

`allowedNextSiblings` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all element information items or wildcards [p.14] that can be inserted as a next sibling of this element, or `null` if this element has no context or schema. Duplicate pairs of {namespaceURI, name} are eliminated.

`allowedParents` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all possible element information items that can appear as a parent this element, or `null` if this element has no context or schema.

`allowedPreviousSiblings` of type `NameList`, readonly
: A `NameList`, as described in [DOM Level 3 Core], of all element information items or wildcards [p.14] that can be inserted as a previous sibling of this element, or `null` if this

element has no context or schema.

`contentType` of type `unsigned short`, readonly

The content type of an element as defined above [p.15] .

`requiredAttributes` of type `NameList`, readonly

A `NameList`, as described in [DOM Level 3 Core], of required attribute information items that must appear on this element, or `null` if this element has no context or schema.

Methods

`canRemoveAttribute`

Verifies if an attribute by the given name can be removed.

Parameters

`attrname` of type `DOMString`

Name of attribute.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canRemoveAttributeNS`

Verifies if an attribute by the given local name and namespace can be removed.

Parameters

`namespaceURI` of type `DOMString`

The namespace URI of the attribute to remove.

`localName` of type `DOMString`

Local name of the attribute to be removed.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canRemoveAttributeNode`

Determines if an attribute node can be removed.

Parameters

`attrNode` of type `Node`

The `Attr` node to remove from the attribute list.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canSetAttribute`

Determines if the value for specified attribute can be set.

Parameters

`attrname` of type `DOMString`

Name of attribute.

`attrval` of type `DOMString`
Value to be assigned to the attribute.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`canSetAttributeNS`

Determines if the attribute with given namespace and qualified name can be created if not already present in the attribute list of the element. If the attribute with the same qualified name and namespaceURI is already present in the element's attribute list, it tests whether the value of the attribute and its prefix can be set to the new value.

Parameters

`namespaceURI` of type `DOMString`
`namespaceURI` of namespace.

`qualifiedName` of type `DOMString`
Qualified name of attribute.

`value` of type `DOMString`
Value to be assigned to the attribute.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`canSetAttributeNode`

Determines if an attribute node can be added.

Parameters

`attrNode` of type `Attr`
`Node` in which the attribute can possibly be set.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`canSetTextContent`

Determines if the text content of this node and its descendants can be set to the string passed in.

Parameters

`possibleTextContent` of type `DOMString`
Possible text content string.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`isElementDefined`

Determines if `name` is defined in the schema. This only applies to global declarations [p.41] . This method is for non-namespace aware schemas.

Parameters

`name` of type `DOMString`

Name of element.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`isElementDefinedNS`

Determines if `name` in this namespace is defined in the current context. Thus not only does this apply to global declarations [p.41] . but depending on the content, this may also apply to local definitions. This method is for namespace aware schemas.

Parameters

`namespaceURI` of type `DOMString`

`namespaceURI` of namespace.

`name` of type `DOMString`

Name of element.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

Interface *CharacterDataEditVAL*

This interface extends the `NodeEditVAL` [p.11] interface with additional methods for document editing. An object implementing this interface must also implement `CharacterData` interface. When validating `CharacterData` nodes, the NodeEditVAL.nodeValidity [p.14] operation must find the nearest parent node in order to do this; if no parent node is found, VAL_UNKNOWN [p.13] is returned. In addition, when VAL_INCOMPLETE [p.12] is passed in as an argument to the `NodeEditVAL.nodeValidity` [p.14] operation to operate on such nodes, the operation considers all the text and not just some of it.

IDL Definition

```
interface CharacterDataEditVAL : NodeEditVAL {
  unsigned short     isWhitespaceOnly();
  unsigned short     canSetData(in DOMString arg);
  unsigned short     canAppendData(in DOMString arg);
  unsigned short     canReplaceData(in unsigned long offset,
                                    in unsigned long count,
                                    in DOMString arg)
                                        raises(DOMException);
  unsigned short     canInsertData(in unsigned long offset,
                                   in DOMString arg)
                                        raises(DOMException);
```

```
  unsigned short     canDeleteData(in unsigned long offset,
                                   in unsigned long count)
                                        raises(DOMException);
};
```

Methods

`canAppendData`

Determines if character data can be appended.

Parameters

`arg` of type `DOMString`

Data to be appended.

Return Value

`unsigned short` A validation state [p.12] constant.

No Exceptions

`canDeleteData`

Determines if character data can be deleted.

Parameters

`offset` of type `unsigned long`

Offset.

`count` of type `unsigned long`

Number of 16-bit units to delete.

Return Value

`unsigned short` A validation state [p.12] constant.

Exceptions

`DOMException` INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

`canInsertData`

Determines if character data can be inserted.

Parameters

`offset` of type `unsigned long`

Offset.

`arg` of type `DOMString`

Argument to be set.

Return Value

`unsigned short` A validation state [p.12] constant.

Exceptions

`DOMException`	INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

`canReplaceData`

Determines if character data can be replaced.

Parameters

`offset` of type `unsigned long`

Offset.

`count` of type `unsigned long`

Replacement.

`arg` of type `DOMString`

Argument to be set.

Return Value

`unsigned short`	A validation state [p.12] constant.

Exceptions

`DOMException`	INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

`canSetData`

Determines if character data can be set.

Parameters

`arg` of type `DOMString`

Argument to be set.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Exceptions

`isWhitespaceOnly`

Determines if character data is only whitespace.

Return Value

`unsigned short`	A validation state [p.12] constant.

No Parameters

No Exceptions

Appendix A: Validation Outcomes

Editor:
Ben Chang, Oracle

A.1 The nodeValidity and validateDocument methods

The following table describes all possible validation outcomes of the `NodeEditVAL.nodeValidity(valType)` [p.14] method.

Validation Type	Validity outcome		
	VAL_TRUE	**VAL_FALSE**	**VAL_UNKNOWN**
`VAL_WF`	The node is well-formed.	The node is not well-formed.	Not applicable.
`VAL_NS_WF`	The node is well-formed. Processor must take into account all the in-scope namespace declarations.	The node is not namespace well-formed. Processor must take into account all the in-scope namespace declarations.	Not applicable.
`VAL_NS_WF`	The node is well-formed. Processor must take into account all namespace declarations in scope.	The node is not namespace well-formed. Processor must take into account all the in-scope namespace declarations.	Not applicable.
`VAL_SCHEMA`	The node is valid: it complies with all the constraints expressed in the schema.	The node fails to comply to all the constraints expressed in the schema.	If the schema is an XML Schema, PSVI **[validity]** property value is unknown.
`VAL_INCOMPLETE`	The node is valid: it complies with the `VAL_INCOMPLETE` definition.	The node is invalid with regard to the `VAL_INCOMPLETE` definition.	If the schema is an XML Schema, PSVI **[validity]** property value is unknown.

The following table describes the outcome of the `DocumentEditVAL.validateDocument()` [p.11] and `NodeEditVAL.nodeValidity(valType)` [p.14] methods, with the latter called on the `DocumentEditVAL` [p.10] node with validationType `NodeEditVAL.VAL_SCHEMA` [p.12] .

Methods	Validity outcome		
	VAL_TRUE	**VAL_FALSE**	**VAL_UNKNOWN**
`validateDocument` and `nodeValidity`, called on the `Document` node with validationType `VAL_SCHEMA`.	If the schema is a DTD, then the document valid constraint is satisfied. If the schema is an XML Schema, then the document validity is the same as the validity of the validation root, i.e., `documentElement`: PSVI **[validity]** valid.	Fails to satisfy the constraints defined.	If the schema is an XML Schema, then schema is not found or the declaration for the validation root is not found: PSVI **[validity]** unknown.

The following table describes outcomes for the `NodeEditVAL.nodeValidity(valType)` [p.14] method called with the validationType `NodeEditVAL.VAL_SCHEMA` [p.12] :

Node types	Validity outcome		
	VAL_TRUE	**VAL_FALSE**	**VAL_UNKNOWN**
`Element`	If the schema is a DTD, then element and attribute validity constraints, including attribute validity constraint defined below are satisfied. If the schema is an XML Schema, then PSVI **[validity]** valid.	Fails to satisfy the constraints defined.	If the schema is an XML Schema, then PSVI **[validity]** unknown.
`Attr`	If the schema is a DTD, then all validity constraints defined in section 3.3.1, "Attribute Type", required and fixed attribute are satisfied. If the schema is an XML Schema, then PSVI **[validity]** valid.	Fails to satisfy the constraints defined.	If the schema is an XML Schema, then PSVI **[validity]** unknown.
`Text`	The node is well-formed.	The node is not well-formed.	If no parent node is found.
`CDATASection`	The node is well-formed.	The node is not well-formed.	If no parent node is found.
`ProcessingInstruction`	The node is well-formed.	The node is not well-formed.	If no parent node is found.
`Comment`	The node is well-formed.	The node is not well-formed.	If no parent node is found.
`EntityReference`	Entity is declared.	Entity is not declared.	Not applicable.
`Entity`	Implementation-specific.	Implementation-specific.	Implementation-specific.
`Notation`	Implementation-specific.	Implementation-specific.	Implementation-specific.
`DocumentType`	Implementation-specific.	Implementation-specific.	Implementation-specific.
`DocumentFragment`	Not applicable.	Not applicable.	Not applicable.

A.2 Other validation operations

The table below describes validation outcomes from `can*` validation operations, such as `NodeEditVAL.canRemoveChild()` [p.13] , or `ElementEditVAL.canSetAttributeNS` [p.18] , `CharacterDataEditVAL.canAppendData()` [p.20] . All these operations attempt to validate with validityType `NodeEditVAL.VAL_INCOMPLETE` [p.12] .

VAL_TRUE	VAL_FALSE	VAL_UNKNOWN
If the associated operation is performed, then the node would be valid with regards to the `VAL_INCOMPLETE` definition or if there is no schema found.	If the associated operation is performed, then the node would be invalid with regards to the `VAL_INCOMPLETE` definition.	Not applicable.

Note: If the document includes more than one type of schema [p.41] , e.g., DTD and XML Schema, and the `DOMConfiguration` "*schema-type*" parameter is not specified, the validation outcome for `NodeEditVAL.VAL_INCOMPLETE` [p.12] and `NodeEditVAL.VAL_SCHEMA` [p.12] is implementation-specific.

Appendix B: IDL Definitions

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

The IDL files are also available as:
http://www.w3.org/TR/2004/REC-DOM-Level-3-Val-20040127/idl.zip

validation.idl:

```
// File: validation.idl

#ifndef _VALIDATION_IDL_
#define _VALIDATION_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module validation
{

  typedef dom::DOMString DOMString;
  typedef dom::DOMStringList DOMStringList;
  typedef dom::Node Node;
  typedef dom::NameList NameList;
  typedef dom::Attr Attr;
  typedef dom::DOMConfiguration DOMConfiguration;

  exception ExceptionVAL {
    unsigned short   code;
  };
  // ExceptionVALCode
  const unsigned short      NO_SCHEMA_AVAILABLE_ERR        = 71;


  interface NodeEditVAL {

    // validationType
    const unsigned short      VAL_WF                         = 1;
    const unsigned short      VAL_NS_WF                      = 2;
    const unsigned short      VAL_INCOMPLETE                 = 3;
    const unsigned short      VAL_SCHEMA                     = 4;


    // validationState
    const unsigned short      VAL_TRUE                       = 5;
    const unsigned short      VAL_FALSE                      = 6;
    const unsigned short      VAL_UNKNOWN                    = 7;

    readonly attribute DOMString       defaultValue;
    readonly attribute DOMStringList   enumeratedValues;
    unsigned short     canInsertBefore(in Node newChild,
                                       in Node refChild);
    unsigned short     canRemoveChild(in Node oldChild);
```

```
  unsigned short     canReplaceChild(in Node newChild,
                                     in Node oldChild);
  unsigned short     canAppendChild(in Node newChild);
  unsigned short     nodeValidity(in unsigned short valType);
};

interface ElementEditVAL : NodeEditVAL {

  // ContentTypeVAL
  const unsigned short      VAL_EMPTY_CONTENTTYPE          = 1;
  const unsigned short      VAL_ANY_CONTENTTYPE            = 2;
  const unsigned short      VAL_MIXED_CONTENTTYPE          = 3;
  const unsigned short      VAL_ELEMENTS_CONTENTTYPE       = 4;
  const unsigned short      VAL_SIMPLE_CONTENTTYPE         = 5;

  readonly attribute NameList        allowedChildren;
  readonly attribute NameList        allowedFirstChildren;
  readonly attribute NameList        allowedParents;
  readonly attribute NameList        allowedNextSiblings;
  readonly attribute NameList        allowedPreviousSiblings;
  readonly attribute NameList        allowedAttributes;
  readonly attribute NameList        requiredAttributes;
  readonly attribute unsigned short  contentType;
  unsigned short     canSetTextContent(in DOMString possibleTextContent);
  unsigned short     canSetAttribute(in DOMString attrname,
                                     in DOMString attrval);
  unsigned short     canSetAttributeNode(in Attr attrNode);
  unsigned short     canSetAttributeNS(in DOMString namespaceURI,
                                       in DOMString qualifiedName,
                                       in DOMString value);
  unsigned short     canRemoveAttribute(in DOMString attrname);
  unsigned short     canRemoveAttributeNS(in DOMString namespaceURI,
                                          in DOMString localName);
  unsigned short     canRemoveAttributeNode(in Node attrNode);
  unsigned short     isElementDefined(in DOMString name);
  unsigned short     isElementDefinedNS(in DOMString namespaceURI,
                                        in DOMString name);
};

interface CharacterDataEditVAL : NodeEditVAL {
  unsigned short     isWhitespaceOnly();
  unsigned short     canSetData(in DOMString arg);
  unsigned short     canAppendData(in DOMString arg);
  unsigned short     canReplaceData(in unsigned long offset,
                                    in unsigned long count,
                                    in DOMString arg)
                                      raises(dom::DOMException);
  unsigned short     canInsertData(in unsigned long offset,
                                   in DOMString arg)
                                      raises(dom::DOMException);
  unsigned short     canDeleteData(in unsigned long offset,
                                   in unsigned long count)
                                      raises(dom::DOMException);
};

interface DocumentEditVAL : NodeEditVAL {
           attribute boolean         continuousValidityChecking;
```

```
                                        // raises(dom::DOMException,
                                        //        ExceptionVAL,
                                        //        dom::DOMException) on setting

    readonly attribute DOMConfiguration domConfig;
    NameList           getDefinedElements(in DOMString namespaceURI);
    unsigned short     validateDocument();
  };
};

#endif // _VALIDATION_IDL_
```

validation.idl:

Appendix C: Java Language Binding

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

The Java files are also available as
http://www.w3.org/TR/2004/REC-DOM-Level-3-Val-20040127/java-binding.zip

org/w3c/dom/validation/ExceptionVAL.java:

```
package org.w3c.dom.validation;

public class ExceptionVAL extends RuntimeException {
    public ExceptionVAL(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // ExceptionVALCode
    public static final short NO_SCHEMA_AVAILABLE_ERR   = 71;

}
```

org/w3c/dom/validation/DocumentEditVAL.java:

```
package org.w3c.dom.validation;

import org.w3c.dom.DOMConfiguration;
import org.w3c.dom.DOMException;
import org.w3c.dom.NameList;

public interface DocumentEditVAL extends NodeEditVAL {
    public boolean getContinuousValidityChecking();
    public void setContinuousValidityChecking(boolean continuousValidityChecking)
 throws DOMException, ExceptionVAL, DOMException;

    public DOMConfiguration getDomConfig();

    public NameList getDefinedElements(String namespaceURI);

    public short validateDocument();

}
```

org/w3c/dom/validation/NodeEditVAL.java:

```
package org.w3c.dom.validation;

import org.w3c.dom.Node;
import org.w3c.dom.DOMStringList;

public interface NodeEditVAL {
    // validationType
```

```
    public static final short VAL_WF                    = 1;
    public static final short VAL_NS_WF                 = 2;
    public static final short VAL_INCOMPLETE            = 3;
    public static final short VAL_SCHEMA                = 4;

    // validationState
    public static final short VAL_TRUE                  = 5;
    public static final short VAL_FALSE                 = 6;
    public static final short VAL_UNKNOWN               = 7;

    public String getDefaultValue();

    public DOMStringList getEnumeratedValues();

    public short canInsertBefore(Node newChild,
                                 Node refChild);

    public short canRemoveChild(Node oldChild);

    public short canReplaceChild(Node newChild,
                                 Node oldChild);

    public short canAppendChild(Node newChild);

    public short nodeValidity(short valType);

}
```

org/w3c/dom/validation/ElementEditVAL.java:

```
package org.w3c.dom.validation;

import org.w3c.dom.Node;
import org.w3c.dom.Attr;
import org.w3c.dom.NameList;

public interface ElementEditVAL extends NodeEditVAL {
    // ContentTypeVAL
    public static final short VAL_EMPTY_CONTENTTYPE     = 1;
    public static final short VAL_ANY_CONTENTTYPE       = 2;
    public static final short VAL_MIXED_CONTENTTYPE     = 3;
    public static final short VAL_ELEMENTS_CONTENTTYPE  = 4;
    public static final short VAL_SIMPLE_CONTENTTYPE    = 5;

    public NameList getAllowedChildren();

    public NameList getAllowedFirstChildren();

    public NameList getAllowedParents();

    public NameList getAllowedNextSiblings();

    public NameList getAllowedPreviousSiblings();

    public NameList getAllowedAttributes();
```

```
    public NameList getRequiredAttributes();

    public short getContentType();

    public short canSetTextContent(String possibleTextContent);

    public short canSetAttribute(String attrname,
                                 String attrval);

    public short canSetAttributeNode(Attr attrNode);

    public short canSetAttributeNS(String namespaceURI,
                                   String qualifiedName,
                                   String value);

    public short canRemoveAttribute(String attrname);

    public short canRemoveAttributeNS(String namespaceURI,
                                      String localName);

    public short canRemoveAttributeNode(Node attrNode);

    public short isElementDefined(String name);

    public short isElementDefinedNS(String namespaceURI,
                                    String name);

}
```

org/w3c/dom/validation/CharacterDataEditVAL.java:

```
package org.w3c.dom.validation;

import org.w3c.dom.DOMException;

public interface CharacterDataEditVAL extends NodeEditVAL {
    public short isWhitespaceOnly();

    public short canSetData(String arg);

    public short canAppendData(String arg);

    public short canReplaceData(int offset,
                                int count,
                                String arg)
                                throws DOMException;

    public short canInsertData(int offset,
                               String arg)
                               throws DOMException;

    public short canDeleteData(int offset,
                               int count)
                               throws DOMException;

}
```

Appendix D: ECMAScript Language Binding

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

Properties of the **ExceptionVAL** Constructor function:

ExceptionVAL.NO_SCHEMA_AVAILABLE_ERR

The value of the constant **ExceptionVAL.NO_SCHEMA_AVAILABLE_ERR** is **71**.

Objects that implement the **ExceptionVAL** interface:

Properties of objects that implement the **ExceptionVAL** interface:

code

This property is a **Number**.

Objects that implement the **DocumentEditVAL** interface:

Objects that implement the **DocumentEditVAL** interface have all properties and functions of the **NodeEditVAL** interface as well as the properties and functions defined below.

Properties of objects that implement the **DocumentEditVAL** interface:

continuousValidityChecking

This property is a **Boolean** and can raise an object that implements the **DOMException** interface or the **ExceptionVAL** interface or the **DOMException** interface on setting.

domConfig

This read-only property is an object that implements the **DOMConfiguration** interface.

Functions of objects that implement the **DocumentEditVAL** interface:

getDefinedElements(namespaceURI)

This function returns an object that implements the **NameList** interface.

The **namespaceURI** parameter is a **String**.

validateDocument()

This function returns a **Number**.

Properties of the **NodeEditVAL** Constructor function:

NodeEditVAL.VAL_WF

The value of the constant **NodeEditVAL.VAL_WF** is **1**.

NodeEditVAL.VAL_NS_WF

The value of the constant **NodeEditVAL.VAL_NS_WF** is **2**.

NodeEditVAL.VAL_INCOMPLETE

The value of the constant **NodeEditVAL.VAL_INCOMPLETE** is **3**.

NodeEditVAL.VAL_SCHEMA

The value of the constant **NodeEditVAL.VAL_SCHEMA** is **4**.

NodeEditVAL.VAL_TRUE

The value of the constant **NodeEditVAL.VAL_TRUE** is **5**.

NodeEditVAL.VAL_FALSE

The value of the constant **NodeEditVAL.VAL_FALSE** is **6**.

NodeEditVAL.VAL_UNKNOWN

The value of the constant **NodeEditVAL.VAL_UNKNOWN** is **7**.

Objects that implement the **NodeEditVAL** interface:

Properties of objects that implement the **NodeEditVAL** interface:

defaultValue

This read-only property is a **String**.

enumeratedValues

This read-only property is an object that implements the **DOMStringList** interface.

Functions of objects that implement the **NodeEditVAL** interface:

canInsertBefore(newChild, refChild)

This function returns a **Number**.

The **newChild** parameter is an object that implements the **Node** interface.

The **refChild** parameter is an object that implements the **Node** interface.

canRemoveChild(oldChild)

This function returns a **Number**.

The **oldChild** parameter is an object that implements the **Node** interface.

canReplaceChild(newChild, oldChild)

This function returns a **Number**.

The **newChild** parameter is an object that implements the **Node** interface.

The **oldChild** parameter is an object that implements the **Node** interface.

canAppendChild(newChild)

This function returns a **Number**.

The **newChild** parameter is an object that implements the **Node** interface.

nodeValidity(valType)

This function returns a **Number**.

The **valType** parameter is a **Number**.

Properties of the **ElementEditVAL** Constructor function:

ElementEditVAL.VAL_EMPTY_CONTENTTYPE

The value of the constant **ElementEditVAL.VAL_EMPTY_CONTENTTYPE** is **1**.

ElementEditVAL.VAL_ANY_CONTENTTYPE

The value of the constant **ElementEditVAL.VAL_ANY_CONTENTTYPE** is **2**.

ElementEditVAL.VAL_MIXED_CONTENTTYPE

The value of the constant **ElementEditVAL.VAL_MIXED_CONTENTTYPE** is **3**.

ElementEditVAL.VAL_ELEMENTS_CONTENTTYPE

The value of the constant **ElementEditVAL.VAL_ELEMENTS_CONTENTTYPE** is **4**.

ElementEditVAL.VAL_SIMPLE_CONTENTTYPE

The value of the constant **ElementEditVAL.VAL_SIMPLE_CONTENTTYPE** is **5**.

Objects that implement the **ElementEditVAL** interface:

Objects that implement the **ElementEditVAL** interface have all properties and functions of the **NodeEditVAL** interface as well as the properties and functions defined below.

Properties of objects that implement the **ElementEditVAL** interface:

allowedChildren

This read-only property is an object that implements the **NameList** interface.

allowedFirstChildren

This read-only property is an object that implements the **NameList** interface.

allowedParents

This read-only property is an object that implements the **NameList** interface.

allowedNextSiblings

This read-only property is an object that implements the **NameList** interface.

allowedPreviousSiblings
This read-only property is an object that implements the **NameList** interface.
allowedAttributes
This read-only property is an object that implements the **NameList** interface.
requiredAttributes
This read-only property is an object that implements the **NameList** interface.
contentType
This read-only property is a **Number**.

Functions of objects that implement the **ElementEditVAL** interface:

canSetTextContent(possibleTextContent)
This function returns a **Number**.
The **possibleTextContent** parameter is a **String**.
canSetAttribute(attrname, attrval)
This function returns a **Number**.
The **attrname** parameter is a **String**.
The **attrval** parameter is a **String**.
canSetAttributeNode(attrNode)
This function returns a **Number**.
The **attrNode** parameter is an object that implements the **Attr** interface.
canSetAttributeNS(namespaceURI, qualifiedName, value)
This function returns a **Number**.
The **namespaceURI** parameter is a **String**.
The **qualifiedName** parameter is a **String**.
The **value** parameter is a **String**.
canRemoveAttribute(attrname)
This function returns a **Number**.
The **attrname** parameter is a **String**.
canRemoveAttributeNS(namespaceURI, localName)
This function returns a **Number**.
The **namespaceURI** parameter is a **String**.
The **localName** parameter is a **String**.
canRemoveAttributeNode(attrNode)
This function returns a **Number**.
The **attrNode** parameter is an object that implements the **Node** interface.
isElementDefined(name)
This function returns a **Number**.
The **name** parameter is a **String**.
isElementDefinedNS(namespaceURI, name)
This function returns a **Number**.
The **namespaceURI** parameter is a **String**.
The **name** parameter is a **String**.

Objects that implement the **CharacterDataEditVAL** interface:

Objects that implement the **CharacterDataEditVAL** interface have all properties and functions of the **NodeEditVAL** interface as well as the properties and functions defined below.

Functions of objects that implement the **CharacterDataEditVAL** interface:

isWhitespaceOnly()
This function returns a **Number**.
canSetData(arg)
This function returns a **Number**.
The **arg** parameter is a **String**.
canAppendData(arg)
This function returns a **Number**.
The **arg** parameter is a **String**.
canReplaceData(offset, count, arg)
This function returns a **Number**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.
The **arg** parameter is a **String**.
This function can raise an object that implements the **DOMException** interface.
canInsertData(offset, arg)
This function returns a **Number**.
The **offset** parameter is a **Number**.
The **arg** parameter is a **String**.
This function can raise an object that implements the **DOMException** interface.
canDeleteData(offset, count)
This function returns a **Number**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.
This function can raise an object that implements the **DOMException** interface.

Appendix E: Acknowledgements

Many people contributed to the DOM specifications (Level 1, 2 or 3), including participants of the DOM Working Group and the DOM Interest Group. We especially thank the following:

Andrew Clover, Andrew Watson (Object Management Group), Andy Heninger (IBM), Angel Diaz (IBM), Arnaud Le Hors (W3C and IBM), Ashok Malhotra (IBM and Microsoft), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), Chris Lovett (Microsoft), Chris Wilson (Microsoft), David Brownell (Sun), David Ezell (Hewlett-Packard Company), David Singer (IBM), Dimitris Dimitriadis (Improve AB and invited expert), Don Park (invited), Elena Litani (IBM), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Jeroen van Rotterdam (X-Hive Corporation), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape/AOL), Jon Ferraiolo (Adobe), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Lauren Wood (SoftQuad Software Inc., *former Chair*), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mary Brady (NIST), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégaret (W3C, *W3C Team Contact and former Chair*), Ramesh Lekshmynarayanan (Merrill Lynch), Ray Whitmer (iMall, Excite@Home, and Netscape/AOL, *Chair*), Rezaur Rahman (Intel), Rich Rollman (Microsoft), Rick Gessner (Netscape), Rick Jelliffe (invited), Rob Relyea (Microsoft), Scott Isaacs (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tim Yu (Oracle), Tom Pixley (Netscape/AOL), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections (Please, keep bugging us with your issues!).

Special thanks to the DOM Conformance Test Suites contributors: Curt Arnold, Fred Drake, Mary Brady (NIST), Rick Rivello (NIST), Robert Clary (Netscape).

E.1 Production Systems

This specification was written in XML. The HTML, OMG IDL, Java and ECMAScript bindings were all produced automatically.

Thanks to Joe English, author of cost, which was used as the basis for producing DOM Level 1. Thanks also to Gavin Nicol, who wrote the scripts which run on top of cost. Arnaud Le Hors and Philippe Le Hégaret maintained the scripts.

After DOM Level 1, we used Xerces as the basis DOM implementation and wish to thank the authors. Philippe Le Hégaret and Arnaud Le Hors wrote the Java programs which are the DOM application.

Thanks also to Jan Kärrman, author of html2ps, which we use in creating the PostScript version of the specification.

Glossary

Editors:
Arnaud Le Hors, W3C
Robert S. Sutor, IBM Research (for DOM Level 1)

Some of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

global declaration
A *global declaration* is a schema declaration, usually for an element or an attribute, that is available for use in content models throughout the schema [p.41] , i.e. a declaration that is not bound to a particular context.

namespace well-formed
A node is a *namespace well-formed* XML node if it is a well-formed [p.41] node, and follows the productions and namespace constraints. If [XML 1.0] is used, the constraints are defined in [XML Namespaces]. If [XML 1.1] is used, the constraints are defined in [XML Namespaces 1.1].

schema
A *schema* defines a set of structural and value constraints applicable to XML documents. Schemas can be expressed in schema languages, such as DTD, XML Schema, etc.

well-formed
A node is a *well-formed* XML node if its serialized form, without doing any transformation during its serialization, matches its respective production in [XML 1.0] or [XML 1.1] (depending on the XML version in use) with all well-formedness constraints related to that production, and if the entities which are referenced within the node are also well-formed. If namespaces for XML are in use, the node must also be namespace well-formed [p.41] .

References

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

The references listed in this section are normatives.

[DOM Level 2 Core]
Document Object Model Level 2 Core Specification, A. Le Hors, et al., Editors. World Wide Web Consortium, 13 November 2000. This version of the DOM Level 2 Core Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113. The latest version of DOM Level 2 Core is available at http://www.w3.org/TR/DOM-Level-2-Core.

[DOM Level 3 Core]
Document Object Model Level 3 Core Specification, A. Le Hors, et al., Editors. World Wide Web Consortium, January 2004. This version of the Document Object Model Level 3 Core Specification is http://www.w3.org/TR/2003/CR-DOM-Level-3-Core-20031107. The latest version of DOM Level 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core.

[ECMAScript]
ECMAScript Language Specification, Third Edition. European Computer Manufacturers Association, Standard ECMA-262, December 1999. This version of the ECMAScript Language is available from http://www.ecma-international.org/.

[Java]
The Java Language Specification, J. Gosling, B. Joy, and G. Steele, Authors. Addison-Wesley, September 1996. Available at http://java.sun.com/docs/books/jls

[OMG IDL]
"OMG IDL Syntax and Semantics" defined in *The Common Object Request Broker: Architecture and Specification, version 2*, Object Management Group. The latest version of CORBA version 2.0 is available at http://www.omg.org/technology/documents/formal/corba_2.htm.

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Editors. World Wide Web Consortium, 10 February 1998, revised 6 October 2000. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version of XML 1.0 is available at http://www.w3.org/TR/REC-xml.

[XML 1.1]
XML 1.1, T. Bray, and al., Editors. World Wide Web Consortium, November 2003. This version of the XML 1.1 Specification is http://www.w3.org/TR/2003/PR-xml11-20031105. The latest version of XML 1.1 is available at http://www.w3.org/TR/xml11.

[XML Namespaces]
Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Editors. World Wide Web Consortium, 14 January 1999. This version of the XML Information Set Recommendation is http://www.w3.org/TR/1999/REC-xml-names-19990114. The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names.

[XML Namespaces 1.1]
Namespaces in XML 1.1, T. Bray, D. Hollander, A. Layman, and R. Tobin, Editors. World Wide Web Consortium, November 2003. This version of the XML Information Set Specification is

http://www.w3.org/TR/2003/PR-xml-names11-20031105/. The latest version of Namespaces in XML is available at http://www.w3.org/TR/xml-names11/.

Index