Document Object Model (DOM) Level 3 Validation Specification

Version 1.0

W3C Working Draft 05 February 2003

This version:
http://www.w3.org/TR/2003/WD-DOM-Level-3-Val-20030205
Latest version:
http://www.w3.org/TR/DOM-Level-3-Val
Previous version:
http://www.w3.org/TR/2002/WD-DOM-Level-3-Val-20021008

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

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

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. The latest status of this document series is maintained at the W3C.

This is the Working Draft specification of "DOM Level 3 Validation". This version is based on the feedback during the last call period. The intention of the Working Group is to ask the Director to move the document to Candidate Recommendation status soon. 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/.

Individuals or organizations are also invited to send a message to the public mailing list if they intend to produce an implementation of this module.

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". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or members of the DOM working group.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM WG members.

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

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

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

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.)

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

[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 APIs to guide construction and editing of XML documents. Examples of such guided editing are queries of the nature that combine questions like "what does the grammar allow me to insert/delete here" and "if I insert/delete here, will the document still be valid."

Users may also want to know several 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 type, in order to help in editing and creating a XML document. Some of these queries would prompt checks and warn you if you’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. To some, 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 sub-tree on demand; and if needed can also require that the document or sub-tree remain valid during this editing process via the `continuousValidityChecking` flag.

A DOM application can use the `hasFeature(feature, version)` method of the `DOMImplementation` interface to determine with parameter values `"VAL-DOC"` and `"3.0"`, respectively, whether or not these interfaces are supported by the implementation.

This chapter focuses on the editing aspects used in the XML document editing world and usage of such information.

1.2. Exceptions

This section describes the "VAL-DOC" exceptions.

Exception *ExceptionVAL*

The Validation operations may throw a `ExceptionVAL` [p.9] as described in their descriptions.

IDL Definition

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

Definition group *ExceptionVALCode*

An integer indicating the type of error generated.

Defined Constants

`NO_GRAMMAR_AVAILABLE_ERR`

If the `DocumentEditVAL` [p.10] related to the node does not have any grammar and `wfValidityCheckLevel` is set to `PARTIAL` or `STRICT_VALIDITY_CHECK`.

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 `CharacterEditVAL` interfaces.

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;
  NameList           getDefinedElementTypes(in DOMString namespaceURI);
  void               validateDocument()
                                          raises(ExceptionVAL);
};
```

Attributes

`continuousValidityChecking` of type `boolean`

An attribute specifying whether continuous checking for the validity of the document is enforced or not. Setting this to `true` will result in an exception being thrown, i.e., `VALIDATION_ERR`, for documents that are invalid at the time of the call. When set to true, the implementation if free to raise the `VALIDATION_ERR` exception on DOM operations that would make the document invalid with respect to "partial validity." If the document is invalid, then this attribute will remain `false`. This attribute is `false` by default.

Methods

`getDefinedElementTypes`

Returns list of all element node names belonging to the element’s namespace. Given the names, nodes can be created from them; note that these are not nodes from the instance document, but rather are new nodes that could be inserted in the document.

Parameters

`namespaceURI` of type `DOMString`

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

Return Value

`NameList`	List of all element node names belonging to the element’s namespace.

No Exceptions

`validateDocument`

Validates the document against the grammar. If the document is mutated during validation, a warning will be issued. In addition, the validation cannot modify the document, e.g., for default attributes. This method makes use of the passed-in error handler, as described in [DOM Level 3 Core] interface.

Exceptions

`ExceptionVAL` [p.9]	NO_GRAMMAR_AVAILABLE_ERR: Raised if an error occurs when the grammar is not available for the document.

No Parameters

No Return Value

Interface *NodeEditVAL*

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

IDL Definition

```
interface NodeEditVAL {

  // CheckTypeVAL
  const unsigned short      WF_CHECK                      = 1;
  const unsigned short      NS_WF_CHECK                   = 2;
  const unsigned short      PARTIAL_VALIDITY_CHECK        = 3;
  const unsigned short      STRICT_VALIDITY_CHECK         = 4;

  readonly attribute DOMString       defaultValue;
  readonly attribute DOMStringList   enumeratedValues;
  boolean            canInsertBefore(in Node newChild,
                                     in Node refChild);
  boolean            canRemoveChild(in Node oldChild);
  boolean            canReplaceChild(in Node newChild,
                                     in Node oldChild);
  boolean            canAppendChild(in Node newChild);
  boolean            isNodeValid(in boolean deep,
                                 in unsigned short wFValidityCheckLevel)
                                        raises(ExceptionVAL);
};
```

Definition group *CheckTypeVAL*

An integer indicating which type of validation this is. Stricter validation for certain XML dialects can be done by extending the `NodeEditVAL` interface adding a new `CheckTypeVAL` constant(s).

Defined Constants

`NS_WF_CHECK`

Check for namespace well-formedness includes `WF_CHECK`.

`PARTIAL_VALIDITY_CHECK`
: Checks for whether this node is *partially valid* [p.33] . It includes `NS_WF_CHECK`.

`STRICT_VALIDITY_CHECK`
: Checks for strict validity of the node with respect to the grammar which by definition includes `NS_WF_CHECK`.

`WF_CHECK`
: Check for well-formedness of this node.

Attributes

`defaultValue` of type `DOMString`, readonly
: The default value specified in an attribute or an element declaration.

`enumeratedValues` of type `DOMStringList`, readonly
: A `DOMStringList` of distinct values for an attribute or an element declaration.

Methods

`canAppendChild`
: Has the same arguments as `Node.appendChild`. Determines whether the `Node.replaceChild` operation would make this document not *partially valid* [p.33] with respect to the grammar.

Parameters

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

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canInsertBefore`
: Determines whether the `Node.insertBefore` operation would make this document not *partially valid* [p.33] with respect to the grammar.

Parameters

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

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

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canRemoveChild`
: Has the same arguments as `Node.removeChild`. Determines whether the `Node.removeChild` operation would make this document not *partially valid* [p.33] with respect to the grammar.

Parameters

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

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canReplaceChild`

Has the same arguments as `Node.replaceChild`. Determines whether the `Node.replaceChild` operation would make this document not *partially valid* [p.33] with respect to the grammar.

Parameters

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

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

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`isNodeValid`

Determines if the Node is valid relative to the grammar. It doesn't normalize before checking if the document is valid. To do so, one would need to explicitly call a normalize method.

Parameters

`deep` of type `boolean`
: Setting the `deep` flag on causes the `isNodeValid` method to check for the whole subtree of the current node for validity. Setting it to `false` only checks the current node and its immediate child nodes. The `validateDocument` method on the `DocumentVAL` interface, however, checks to determine whether the entire document is valid.

`wFValidityCheckLevel` of type `unsigned short`
: Flag to tell at what level validity and well-formedness checking is done.

Return Value

`boolean`	`true` if the node is valid/well-formed in the grammar and check level defined by `wfValidityCheckLevel`, `false` if not.

Exceptions

`ExceptionVAL` [p.9]	NO_GRAMMAR_AVAILABLE_ERR: Exception is raised if the DocumentEditVAL related to this node does not have any grammar associated with it and `wfValidityCheckLevel` is set to `PARTIAL` or `STRICT_VALIDITY_CHECK`.

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 `Element` interface.

IDL Definition

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

Attributes

`allowedAttributes` of type `NameList`, readonly
: A `NameList` of possible `Attr` nodes that can appear with this type of element. Given the names, nodes can be created from them; note that these are not nodes from the instance document, but rather are new nodes that could be inserted in the document.

`allowedChildren` of type `NameList`, readonly
: A `NameList` of possible `Element` nodes that can appear as children of this type of element. Note that if no context of this element exists, then this is `NULL`; it is an empty list if the element is not in the document tree. Given the names, nodes can be created from them; note that these are not nodes from the instance document, but rather are new nodes that could be inserted in the document.

`allowedNextSiblings` of type `NameList`, readonly
: A `NameList` of possible sibling `Element` nodes that can appear after this element.

`allowedParents` of type `NameList`, readonly
: A `NameList` of possible `Element` nodes that can appear as a parent of this type of element. Note that if no context of this element exists, for example, the parent element of

this element, then this is `NULL`; it is an empty list if the element is not in the document tree. Given the names, nodes can be created from them; note that these are not nodes from the instance document, but rather are new nodes that could be inserted in the document.

`allowedPreviousSiblings` of type `NameList`, readonly

A `NameList` of possible sibling `Element` nodes that can appear before this element.

`requiredAttributes` of type `NameList`, readonly

A `NameList` of required `Attr` nodes that must appear with this type of element. Given the names, nodes can be created from them; note that these are not nodes from the instance document, but rather are new nodes that could be inserted in the document.

Methods

`canRemoveAttribute`

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

Parameters

`attrname` of type `DOMString`

Name of attribute.

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

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

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

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

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

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

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

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. See DOM core `setAttributeNS`.
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

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canSetAttributeNode`
Determines if an attribute node can be added with respect to the validity check level.
Parameters
`attrNode` of type `Attr`
`Node` in which the attribute can possibly be set.
Return Value

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`contentType`
Determines element content type.
Return Value

`unsigned short`	Constant for one of `EMPTY_CONTENTTYPE` - content type of an element which has neither child elements nor character data, `ANY_CONTENTTYPE` - content type of an element which can contain zero or more child elements as well as character data, `MIXED_CONTENTTYPE` - content type of an element which contains character data optionally interspersed with child elements, `ELEMENTS_CONTENTTYPE` - content type of an element which contains only child elements optionally separated by whitespace, `SIMPLE_CONTENTTYPE` - content type of an element which contains character data with attributes.

No Parameters

No Exceptions

`isElementDefined`

Determines if `name` is defined in the grammar.

Parameters

`name` of type `DOMString`

Name of element.

Return Value

`boolean`	A boolean that is `true` if the element is defined, false otherwise.

No Exceptions

`isElementDefinedNS`

Determines if `name` in this namespace is defined in the current context.

Parameters

`namespaceURI` of type `DOMString`

`namespaceURI` of namespace.

`name` of type `DOMString`

Name of element.

Return Value

`boolean`	A boolean that is `true` if the element is defined, false otherwise.

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.

IDL Definition

```
interface CharacterDataEditVAL : NodeEditVAL {
  readonly attribute boolean         isWhitespaceOnly;
  boolean            canSetData(in DOMString arg);
  boolean            canAppendData(in DOMString arg);
```

```
  boolean            canReplaceData(in unsigned long offset,
                                    in unsigned long count,
                                    in DOMString arg);
  boolean            canInsertData(in unsigned long offset,
                                   in DOMString arg);
  boolean            canDeleteData(in unsigned long offset,
                                   in unsigned long count);
};
```

Attributes

`isWhitespaceOnly` of type `boolean`, readonly

`true` if content only whitespace; `false` for non-whitespace.

Methods

`canAppendData`

Determines if data can be appended.

Parameters

`arg` of type `DOMString`

Data to be appended.

Return Value

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canDeleteData`

Determines if 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

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canInsertData`

Determines if data can be inserted.

Parameters

`offset` of type `unsigned long`

Offset.

`arg` of type `DOMString`

Argument to be set.

Return Value

`boolean` `true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canReplaceData`

Determines if 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

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

`canSetData`

Determines if data can be set.

Parameters

`arg` of type `DOMString`

Argument to be set.

Return Value

`boolean`	`true` if no reason it can't be done; `false` if it can't be done.

No Exceptions

1.4. Range-related Document-Editing Interfaces

This section contains Range-related "Document-editing" methods.

Interface *RangeVAL*

This interface extends the `Range` interface with additional methods for guided document editing.

IDL Definition

```
interface RangeVAL : Range {
  boolean            canSurroundContents(in Node node1,
                                         in Node node2,
                                         in Node b);
  NameList           getAlternativeElements(in Node refChild);
};
```

Methods

`canSurroundContents`

Determines if a passed-in node can be "surrounded" by two other nodes. For example, this convenience method can: 1) get all children of "x", 2) get all possible parents of "y", and

then 3) check if the passed-in node "b" is present in both sets.

Parameters

`node1` of type `Node`

First node to "surround" passed-in node "b".

`node2` of type `Node`

Second node to "surround" passed-in node "b".

`b` of type `Node`

Node to be "surrounded" by.

Return Value

`boolean`	`true` if it is; `false` if it is not.

No Exceptions

`getAlternativeElements`

Returns a list of names of alternative elements, given a reference child as a parameter.

Parameters

`refChild` of type `Node`

Reference child.

Return Value

`NameList`	A list of names of alternative elements.

No Exceptions

Appendix A: 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/2003/WD-DOM-Level-3-Val-20030205/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::Range Range;

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


  interface NodeEditVAL {

    // CheckTypeVAL
    const unsigned short      WF_CHECK                       = 1;
    const unsigned short      NS_WF_CHECK                    = 2;
    const unsigned short      PARTIAL_VALIDITY_CHECK         = 3;
    const unsigned short      STRICT_VALIDITY_CHECK          = 4;

    readonly attribute DOMString       defaultValue;
    readonly attribute DOMStringList   enumeratedValues;
    boolean            canInsertBefore(in Node newChild,
                                       in Node refChild);
    boolean            canRemoveChild(in Node oldChild);
    boolean            canReplaceChild(in Node newChild,
                                       in Node oldChild);
    boolean            canAppendChild(in Node newChild);
    boolean            isNodeValid(in boolean deep,
                                   in unsigned short wFValidityCheckLevel)
                                        raises(ExceptionVAL);
  };
```

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

  interface CharacterDataEditVAL : NodeEditVAL {
    readonly attribute boolean         isWhitespaceOnly;
    boolean            canSetData(in DOMString arg);
    boolean            canAppendData(in DOMString arg);
    boolean            canReplaceData(in unsigned long offset,
                                      in unsigned long count,
                                      in DOMString arg);
    boolean            canInsertData(in unsigned long offset,
                                     in DOMString arg);
    boolean            canDeleteData(in unsigned long offset,
                                     in unsigned long count);
  };

  interface DocumentEditVAL : NodeEditVAL {
             attribute boolean         continuousValidityChecking;
    NameList           getDefinedElementTypes(in DOMString namespaceURI);
    void               validateDocument()
                                        raises(ExceptionVAL);
  };

  interface RangeVAL : Range {
    boolean            canSurroundContents(in Node node1,
                                           in Node node2,
                                           in Node b);
    NameList           getAlternativeElements(in Node refChild);
  };
};

#endif // _VALIDATION_IDL_
```

Appendix B: 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/2003/WD-DOM-Level-3-Val-20030205/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_GRAMMAR_AVAILABLE_ERR  = 71;

}
```

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

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

import org.w3c.dom.NameList;

public interface DocumentEditVAL extends NodeEditVAL {
    public boolean getContinuousValidityChecking();
    public void setContinuousValidityChecking(boolean continuousValidityChecking);

    public NameList getDefinedElementTypes(String namespaceURI);

    public void validateDocument()
                                throws ExceptionVAL;

}
```

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

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

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

public interface NodeEditVAL {
    // CheckTypeVAL
    public static final short WF_CHECK                  = 1;
    public static final short NS_WF_CHECK               = 2;
    public static final short PARTIAL_VALIDITY_CHECK    = 3;
    public static final short STRICT_VALIDITY_CHECK     = 4;
```

```
    public String getDefaultValue();

    public DOMStringList getEnumeratedValues();

    public boolean canInsertBefore(Node newChild,
                                   Node refChild);

    public boolean canRemoveChild(Node oldChild);

    public boolean canReplaceChild(Node newChild,
                                   Node oldChild);

    public boolean canAppendChild(Node newChild);

    public boolean isNodeValid(boolean deep,
                               short wFValidityCheckLevel)
                               throws ExceptionVAL;

}
```

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 {
    public NameList getAllowedChildren();

    public NameList getAllowedParents();

    public NameList getAllowedNextSiblings();

    public NameList getAllowedPreviousSiblings();

    public NameList getAllowedAttributes();

    public NameList getRequiredAttributes();

    public short contentType();

    public boolean canSetAttribute(String attrname,
                                   String attrval);

    public boolean canSetAttributeNode(Attr attrNode);

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

    public boolean canRemoveAttribute(String attrname);

    public boolean canRemoveAttributeNS(String namespaceURI,
```

```
                                        String localName);

    public boolean canRemoveAttributeNode(Node attrNode);

    public boolean isElementDefined(String name);

    public boolean isElementDefinedNS(String namespaceURI,
                                      String name);

}
```

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

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

public interface CharacterDataEditVAL extends NodeEditVAL {
    public boolean getIsWhitespaceOnly();

    public boolean canSetData(String arg);

    public boolean canAppendData(String arg);

    public boolean canReplaceData(int offset,
                                  int count,
                                  String arg);

    public boolean canInsertData(int offset,
                                 String arg);

    public boolean canDeleteData(int offset,
                                 int count);

}
```

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

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

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

public interface RangeVAL extends Range {
    public boolean canSurroundContents(Node node1,
                                       Node node2,
                                       Node b);

    public NameList getAlternativeElements(Node refChild);

}
```

Appendix C: 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_GRAMMAR_AVAILABLE_ERR

The value of the constant **ExceptionVAL.NO_GRAMMAR_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**.

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

getDefinedElementTypes(namespaceURI)

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

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

validateDocument()

This function has no return value.

This function can raise an object that implements the **ExceptionVAL** interface.

Properties of the **NodeEditVAL** Constructor function:

NodeEditVAL.WF_CHECK

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

NodeEditVAL.NS_WF_CHECK

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

NodeEditVAL.PARTIAL_VALIDITY_CHECK

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

NodeEditVAL.STRICT_VALIDITY_CHECK

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

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 **Boolean**.

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 **Boolean**.

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

canReplaceChild(newChild, oldChild)

This function returns a **Boolean**.

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 **Boolean**.

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

isNodeValid(deep, wFValidityCheckLevel)

This function returns a **Boolean**.

The **deep** parameter is a **Boolean**.

The **wFValidityCheckLevel** parameter is a **Number**.

This function can raise an object that implements the **ExceptionVAL** interface.

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.

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.

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

contentType()

This function returns a **Number**.

canSetAttribute(attrname, attrval)

This function returns a **Boolean**.

The **attrname** parameter is a **String**.

The **attrval** parameter is a **String**.

canSetAttributeNode(attrNode)

This function returns a **Boolean**.

The **attrNode** parameter is an object that implements the **Attr** interface.

canSetAttributeNS(namespaceURI, qualifiedName, value)

This function returns a **Boolean**.

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

The **qualifiedName** parameter is a **String**.

The **value** parameter is a **String**.

canRemoveAttribute(attrname)
This function returns a **Boolean**.
The **attrname** parameter is a **String**.
canRemoveAttributeNS(namespaceURI, localName)
This function returns a **Boolean**.
The **namespaceURI** parameter is a **String**.
The **localName** parameter is a **String**.
canRemoveAttributeNode(attrNode)
This function returns a **Boolean**.
The **attrNode** parameter is an object that implements the **Node** interface.
isElementDefined(name)
This function returns a **Boolean**.
The **name** parameter is a **String**.
isElementDefinedNS(namespaceURI, name)
This function returns a **Boolean**.
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.

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

isWhitespaceOnly
This read-only property is a **Boolean**.

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

canSetData(arg)
This function returns a **Boolean**.
The **arg** parameter is a **String**.
canAppendData(arg)
This function returns a **Boolean**.
The **arg** parameter is a **String**.
canReplaceData(offset, count, arg)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.
The **arg** parameter is a **String**.
canInsertData(offset, arg)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **arg** parameter is a **String**.
canDeleteData(offset, count)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.

Objects that implement the **RangeVAL** interface:

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

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

canSurroundContents(node1, node2, b)

This function returns a **Boolean**.

The **node1** parameter is an object that implements the **Node** interface.

The **node2** parameter is an object that implements the **Node** interface.

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

getAlternativeElements(refChild)

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

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

Appendix D: Acknowledgements

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

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), 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!).

D.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)

Several 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.

document element
There is only one document element in a `Document`. This element node is a child of the `Document` node. See *Well-Formed XML Documents* in XML [XML 1.0].

document order
There is an ordering, *document order*, defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities. Thus, the *document element* [p.33] node will be the first node. Element nodes occur before their children. Thus, document order orders element nodes in order of the occurrence of their start-tag in the XML (after expansion of entities). The attribute nodes of an element occur after the element and before its children. The relative order of attribute nodes is implementation-dependent.

event
An event is the representation of some asynchronous occurrence (such as a mouse click on the presentation of the element, or the removal of child node from an element, or any of unthinkably many other possibilities) that gets associated with an *event target* [p.33] .

event target
The object to which an *event* [p.33] is targeted.

partially valid
A node in a DOM tree is *partially valid* if it is *well formed* [p.33] (this part is for comments and processing instructions) and its immediate children are those expected by the content model. The node may be missing trailing required children yet still be considered *partially valid*.

target node
The target node is the node representing the *event target* [p.33] to which an *event* [p.33] is targeted using the DOM event flow defined in [DOM Level 3 Events].

tokenized
The description given to various information items (for example, attribute values of various types, but not including the StringType CDATA) after having been processed by the XML processor. The process includes stripping leading and trailing white space, and replacing multiple space characters by one. See the definition of tokenized type.

well-formed
A node is a *well-formed* XML node if it matches its respective production in [XML 1.0], meets all well-formedness constraints related to the production, if the entities which are referenced within the node are also well-formed. See also the definition for *well-formed* XML documents in [XML 1.0].

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 3 Core]
Document Object Model Level 3 Core Specification, A. Le Hors, et al., Editors. World Wide Web Consortium, October 2002. This version of the Document Object Model Level 3 Core Specification is http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20021022. The latest version of DOM Level 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core.

[DOM Level 3 Events]
Document Object Model Level 3 Events Specification, P. Le Hégaret, T. Pixley, Editors. World Wide Web Consortium, July 2002. This version of the Document Object Model Level 3 Events Specification is http://www.w3.org/TR/DOM-Level-3-Events. The latest version of Document Object Model Level 3 Events is available at http://www.w3.org/TR/DOM-Level-3-Events.

[ECMAScript]
ECMAScript Language Specification, Third Edition. European Computer Manufacturers Association, December 1999. This version of the ECMAScript Language is available from http://www.ecma.ch/.

[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.

Index