Document Object Model (DOM) Level 3 Load and Save Specification

Version 1.0

W3C Working Draft 25 July 2002

This version:
http://www.w3.org/TR/2002/WD-DOM-Level-3-LS-20020725
Latest version:
http://www.w3.org/TR/DOM-Level-3-LS
Previous version:
http://www.w3.org/TR/2002/WD-DOM-Level-3-ASLS-20020409

Editors:
Johnny Stenback, *Netscape*
Andy Heninger, *IBM (until March 2001)*

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

Copyright ©2002 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

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

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 a W3C Working Draft for review by W3C members and other interested parties. This DOM module has been dissociated from the Abstract Schema DOM module since the modules were independent and the latter is no longer a work in progress.

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.

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

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

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 3
Copyright Notice 5

1. Document Object Model Load and Save 9

Appendix A: IDL Definitions 51
Appendix B: Java Language Binding 55
Appendix C: ECMAScript Language Binding 61
Appendix D: Acknowledgements 65
Glossary 67
References 69
Index 73

Expanded Table of Contents

Expanded Table of Contents 3
Copyright Notice 5
W3C Document Copyright Notice and License 5
W3C Software Copyright Notice and License 6

1. Document Object Model Load and Save 9
1.1. Overview 9
1.1.1. Overview of the Interfaces 9
1.1.2. The DOMInputStream type 9
1.1.3. The DOMOutputStream type 10
1.1.4. The DOMReader type 10
1.2. Fundamental interfaces 10
1.3. Load Interfaces 15
1.4. Save Interfaces 28
1.5. Issue List 35
1.5.1. Open Issues 36
1.5.2. Resolved Issues 36

Appendix A: IDL Definitions 51
Appendix B: Java Language Binding 55
Appendix C: ECMAScript Language Binding 61
Appendix D: Acknowledgements 65
D.1. Production Systems 65
Glossary 67
References 69
1. Normative references 69
2. Informative references 70
Index 73

Expanded Table of Contents

Copyright Notice

Copyright © 2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

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/copyright-documents-19990405.

Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

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

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

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

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

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

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/copyright-software-19980720

Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

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

This W3C work (including software, documents, 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 use, copy, and modify 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, that you make:

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. If none exist, then a notice of the following form: "Copyright © [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/."

3. Notice of any changes or modifications to the W3C 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 Software Copyright Notice and License

1. Document Object Model Load and Save

Editors:
Johnny Stenback, Netscape
Andy Heninger, IBM (until March 2001)

1.1. Overview

This section defines a set of interfaces for loading and saving document objects as defined in [DOM Level 3 Core]. The functionality specified in this section (the *Load and Save* functionality) is sufficient to allow software developers and web script authors to load and save XML content inside conforming products. The DOM Load and Save *API* [p.67] also allows filtering of XML content using only DOM API calls; access and manipulation of the `Document` is defined in [DOM Level 3 Core].

The proposal for loading is influenced by the Java APIs for XML Processing [JAXP] and by SAX2 [SAX].

1.1.1. Overview of the Interfaces

The list of interfaces involved with the Loading and Saving XML documents is:

- `DOMImplementationLS` [p.10] -- A new `DOMImplementation` interface that provides the factory methods for creating the objects required for loading and saving.
- `DOMBuilder` [p.22] -- A parser interface.
- `DOMInputSource` [p.15] -- Encapsulate information about the XML document to be loaded.
- `DOMEntityResolver` [p.18] -- During loading, provides a way for applications to redirect references to external entities.
- `DOMBuilderFilter` [p.19] -- Provide the ability to examine and optionally remove Element nodes as they are being processed during the parsing of a document.
- `DOMWriter` [p.28] -- An interface for writing out or serializing DOM documents.
- `DocumentLS` [p.13] -- Provides a client or browser style interface for loading and saving.
- `ParseErrorEvent` -- ParseErrorEvent is the event that is fired if there's an error in the XML document being parsed using the methods of DocumentLS.

1.1.2. The DOMInputStream type

To ensure interoperability, the DOM Load and Save specifies the following:

Type Definition *DOMInputStream*

A `DOMInputStream` [p.9] represents a reference to a byte stream source of an XML input.

IDL Definition

```
typedef Object DOMInputStream;
```

Note: Even though the DOM uses the type `DOMInputStream` [p.9] , bindings may use different types. For example, in Java `DOMInputStream` is bound to the `java.io.InputStream` type, while in ECMAScript `DOMInputStream` is bound to `Object`.

1.1.3. The DOMOutputStream type

To ensure interoperability, the DOM Load and Save specifies the following:

Type Definition *DOMOutputStream*

A `DOMOutputStream` [p.10] represents a byte stream destination for the XML output.

IDL Definition

```
typedef Object DOMOutputStream;
```

Note: Even though the DOM uses the type `DOMOutputStream` [p.10] , bindings may use different types. For example, in Java `DOMOutputStream` is bound to the `java.io.OutputStream` type, while in ECMAScript `DOMOutputStream` is bound to `Object`.

1.1.4. The DOMReader type

To ensure interoperability, the DOM Load and Save specifies the following:

Type Definition *DOMReader*

A `DOMReader` [p.10] represents a character stream for the XML input.

IDL Definition

```
typedef Object DOMReader;
```

Note: Even though the DOM uses the type `DOMReader` [p.10] , bindings may use different types. For example, in Java `DOMReader` is bound to the `java.io.Reader` type, while in ECMAScript `DOMReader` is *NOT* bound, and therefore as no recommended meaning in ECMAScript.

1.2. Fundamental interfaces

The interface within this section is considered fundamental, and must be fully implemented by all conforming implementations of the DOM Load and Save module.

Interface *DOMImplementationLS*

`DOMImplementationLS` contains the factory methods for creating objects that implement the `DOMBuilder` [p.22] (parser) and `DOMWriter` [p.28] (serializer) interfaces.

The expectation is that an instance of the DOMImplementationLS interface can be obtained by using binding-specific casting methods on an instance of the DOMImplementation interface or, if the Document supports the feature "Core" version "3.0" defined in [DOM Level 3 Core], by using the method Node.getInterface with parameter values "LS-Load" and "3.0" (respectively).

IDL Definition

```
interface DOMImplementationLS {

  // DOMIMplementationLSMode
  const unsigned short      MODE_SYNCHRONOUS               = 1;
  const unsigned short      MODE_ASYNCHRONOUS              = 2;

  DOMBuilder         createDOMBuilder(in unsigned short mode,
                                      in DOMString schemaType)
                                        raises(DOMException);
  DOMWriter          createDOMWriter();
  DOMInputSource     createDOMInputSource();
};
```

Definition group *DOMIMplementationLSMode*

An integer indicating which type of mode this is.

Defined Constants

MODE_ASYNCHRONOUS
: Create an asynchronous DOMBuilder [p.22] .

MODE_SYNCHRONOUS
: Create a synchronous DOMBuilder [p.22] .

Methods

createDOMBuilder
: Create a new DOMBuilder [p.22] . The newly constructed parser may then be configured by means of its setFeature method, and used to parse documents by means of its parse method.

Parameters

mode of type unsigned short
: The mode argument is either MODE_SYNCHRONOUS or MODE_ASYNCHRONOUS, if mode is MODE_SYNCHRONOUS then the DOMBuilder [p.22] that is created will operate in synchronous mode, if it's MODE_ASYNCHRONOUS then the DOMBuilder that is created will operate in asynchronous mode.

schemaType of type DOMString
: An absolute URI representing the type of the schema language used during the load of a Document using the newly created DOMBuilder [p.22] . Note that no lexical checking is done on the absolute URI. In order to create a DOMBuilder for any kind of schema types (i.e. the DOMBuilder will be free to use any schema found), use the value null.

Note: For W3C XML Schema [XML Schema Part 1], applications must use the value `"http://www.w3.org/2001/XMLSchema"`. For XML DTD [XML 1.0], applications must use the value `"http://www.w3.org/TR/REC-xml"`. Other Schema languages are outside the scope of the W3C and therefore should recommend an absolute URI in order to use this method.

Return Value

`DOMBuilder` [p.22]	The newly created `DOMBuilder` object. This `DOMBuilder` is either synchronous or asynchronous depending on the value of the `mode` argument. **Note:** By default, the newly created `DOMBuilder` does not contain a `DOMErrorHandler`, i.e. the value of the `errorHandler` is `null`. However, implementations may provide a default error handler at creation time. In that case, the initial value of the `errorHandler` attribute on the new created `DOMBuilder` contains a reference to the default error handler.

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if the requested mode or schema type is not supported.

`createDOMInputSource`

Create a new "empty" `DOMInputSource` [p.15] .

Return Value

`DOMInputSource` [p.15]	The newly created `DOMInputSource` object.

No Parameters

No Exceptions

`createDOMWriter`

Create a new `DOMWriter` [p.28] object. `DOMWriters` are used to serialize a DOM tree back into an XML document.

Return Value

`DOMWriter` [p.28]	The newly created `DOMWriter` object. **Note:** By default, the newly created `DOMWriter` does not contain a `DOMErrorHandler`, i.e. the value of the `errorHandler` is `null`. However, implementations may provide a default error handler at creation time. In that case, the initial value of the `errorHandler` attribute on the new created `DOMWriter` contains a reference to the default error handler.

No Parameters
No Exceptions

Interface *DocumentLS*

The `DocumentLS` interface provides a mechanism by which the content of a document can be replaced with the DOM tree produced when loading a URI, or parsing a string. The expectation is that an instance of the `DocumentLS` interface can be obtained by using binding-specific casting methods on an instance of the `Document` interface or, if the `Document` supports the feature `"Core"` version `"3.0"` defined in [DOM Level 3 Core], by using the method `Node.getInterface` with parameter values `"LS-Load"` and `"3.0"` (respectively).

IDL Definition

```
interface DocumentLS {
           attribute boolean         async;
                                        // raises(DOMException) on setting

  void               abort();
  boolean            load(in DOMString uri);
  boolean            loadXML(in DOMString source);
  DOMString          saveXML(in Node snode)
                                        raises(DOMException);
};
```

Attributes

`async` of type `boolean`

Indicates whether the method load should be synchronous or asynchronous. When the async attribute is set to `true` the load method returns control to the caller before the document has completed loading. The default value of this attribute is `false`.

Issue async-1:

Should the DOM spec define the default value of this attribute? What if implementing both async and sync IO is impractical in some systems?

Resolution: 2001-09-14. default is `false` but we need to check with Mozilla and IE.

Exceptions on setting

`DOMException`	NOT_SUPPORTED_ERR: Raised if the implementation doesn't support the mode the attribute is being set to.

Methods

`abort`

If the document is currently being loaded as a result of the method `load` being invoked the loading and parsing is immediately aborted. The possibly partial result of parsing the document is discarded and the document is cleared.

No Parameters
No Return Value
No Exceptions

`load`

Replaces the content of the document with the result of parsing the given URI. Invoking this method will either block the caller or return to the caller immediately depending on the

value of the async attribute. Once the document is fully loaded the document will fire a "load" event that the caller can register as a listener for. If an error occurs the document will fire an "error" event so that the caller knows that the load failed (see `ParseErrorEvent`). If this method is called on a document that is currently loading, the current load is interrupted and the new URI load is initiated.
When invoking this method the features used in the `DOMBuilder` [p.22] interface are assumed to have their default values with the exception that the feature "entities" is "false".

Parameters

`uri` of type `DOMString`

The URI reference for the XML file to be loaded. If this is a relative URI, the base URI used by the implementation is implementation dependent.

Return Value

`boolean`	If async is set to `true` `load` returns `true` if the document load was successfully initiated. If an error occurred when initiating the document load `load` returns `false`. If async is set to `false` `load` returns `true` if the document was successfully loaded and parsed. If an error occurred when either loading or parsing the URI `load` returns `false`.

No Exceptions

`loadXML`

Replace the content of the document with the result of parsing the input string, this method is always synchronous. This method always parses from a DOMString, which means the data is always UTF16. All other encoding information is ignored.
The features used in the `DOMBuilder` [p.22] interface are assumed to have their default values when invoking this method.

Parameters

`source` of type `DOMString`

A string containing an XML document.

Return Value

`boolean`	`true` if parsing the input string succeeded without errors, otherwise `false`.

No Exceptions

`saveXML`

Save the document or the given node and all its descendants to a string (i.e. serialize the document or node).
The features used in the `DOMWriter` [p.28] interface are assumed to have their default values when invoking this method.

Parameters

`snode` of type `Node`

Specifies what to serialize, if this parameter is `null` the whole document is serialized, if it's non-null the given node is serialized.

Return Value

`DOMString`	The serialized document or `null` in case an error occured.

Exceptions

`DOMException`	WRONG_DOCUMENT_ERR: Raised if the node passed in as the node parameter is from an other document.

1.3. Load Interfaces

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "LS-Load" and "3.0" (respectively) to determine whether or not these interfaces are supported by the implementation. In order to fully support them, an implementation must also support the "Core" feature defined in the DOM Level 3 Core specification [DOM Level 3 Core].

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "LS-Load-Async" and "3.0" (respectively) to determine whether or not the asynchronous mode is supported by the implementation. In order to fully support the asyncrhonous mode, an implementation must also support the "LS-Load" feature defined in this section.

Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core].

Interface *DOMInputSource*

This interface represents a single input source for an XML entity.

This interface allows an application to encapsulate information about an input source in a single object, which may include a public identifier, a system identifier, a byte stream (possibly with a specified encoding), and/or a character stream.

The exact definitions of a byte stream and a character stream are binding dependent.

There are two places that the application will deliver this input source to the parser: as the argument to the `parse` method, or as the return value of the `DOMEntityResolver.resolveEntity` [p.18] method.

(***ED:*** There are at least three places where DOMInputSource is passed to the parser (parseWithContext).)

The `DOMBuilder` [p.22] will use the `DOMInputSource` object to determine how to read XML input. If there is a character stream available, the parser will read that stream directly; if not, the parser will use a byte stream, if available; if neither a character stream nor a byte stream is available,

the parser will attempt to open a URI connection to the resource identified by the system identifier.

A `DOMInputSource` object belongs to the application: the parser shall never modify it in any way (it may modify a copy if necessary).

Note: Even though all attributes in this interface are writable the DOM implementation is expected to never mutate a DOMInputSource.

IDL Definition

```
interface DOMInputSource {
           attribute DOMInputStream  byteStream;
           attribute DOMReader       characterStream;
           attribute DOMString       stringData;
           attribute DOMString       encoding;
           attribute DOMString       publicId;
           attribute DOMString       systemId;
           attribute DOMString       baseURI;
};
```

Attributes

`baseURI` of type `DOMString`
: The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving relative URIs to absolute URIs. If the baseURI is itself a relative URI, the behavior is implementation dependent.

`byteStream` of type `DOMInputStream` [p.9]
: An attribute of a language-binding dependent type that represents a stream of bytes.
The parser will ignore this if there is also a character stream specified, but it will use a byte stream in preference to opening a URI connection itself.
If the application knows the character encoding of the byte stream, it should set the encoding attribute. Setting the encoding in this way will override any encoding specified in the XML declaration itself.

`characterStream` of type `DOMReader` [p.10]
: An attribute of a language-binding dependent type that represents a stream of *16-bit units.* [p.67] Application must encode the stream using UTF-16 (defined in [Unicode 2.0] and Amendment 1 of [ISO/IEC 10646]).
If a character stream is specified, the parser will ignore any byte stream and will not attempt to open a URI connection to the system identifier.

`encoding` of type `DOMString`
: The character encoding, if known. The encoding must be a string acceptable for an XML encoding declaration ([XML 1.0] section 4.3.3 "Character Encoding in Entities").
This attribute has no effect when the application provides a character stream. For other sources of input, an encoding specified by means of this attribute will override any encoding specified in the XML declaration or the Text declaration, or an encoding obtained from a higher level protocol, such as HTTP [IETF RFC 2616].

`publicId` of type `DOMString`
: The public identifier for this input source. The public identifier is always optional: if the application writer includes one, it will be provided as part of the location information.

`stringData` of type `DOMString`
: A string attribute that represents a sequence of 16 bit units (utf-16 encoded characters).
If string data is available in the input source, the parser will ignore the character stream and the byte stream and will not attempt to open a URI connection to the system identifier.

`systemId` of type `DOMString`
: The system identifier, a URI reference [IETF RFC 2396], for this input source. The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to fetch the ressource identifier by the URI reference only if there is no byte stream or character stream specified).
If the application knows the character encoding of the object pointed to by the system identifier, it can register the encoding by setting the encoding attribute.
If the system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the behavior is implementation dependent.

Interface *LSLoadEvent*

This interface represents a load event object that signals the completion of a document load.

IDL Definition

```
interface LSLoadEvent : events::Event {
  readonly attribute Document        newDocument;
  readonly attribute DOMInputSource  inputSource;
};
```

Attributes

`inputSource` of type `DOMInputSource` [p.15] , readonly
: The input source that was parsed.

`newDocument` of type `Document`, readonly
: The document that finished loading.

Interface *LSProgressEvent*

This interface represents a progress event object that notifies the application about progress as a document is parsed. It extends the `Event` interface defined in [DOM Level 3 Events].

IDL Definition

```
interface LSProgressEvent : events::Event {
  readonly attribute DOMInputSource  inputSource;
  readonly attribute unsigned long   position;
  readonly attribute unsigned long   totalSize;
};
```

Attributes

`inputSource` of type `DOMInputSource` [p.15] , readonly
: The input source that is being parsed.

`position` of type `unsigned long`, readonly
: The current position in the input source, including all external entities and other resources that have been read.

`totalSize` of type `unsigned long`, readonly
The total size of the document including all external resources, this number might change as a document is being parsed if references to more external resources are seen.

Interface *DOMEntityResolver*

`DOMEntityResolver` Provides a way for applications to redirect references to external entities.

Applications needing to implement customized handling for external entities must implement this interface and register their implementation by setting the `entityResolver` attribute of the `DOMBuilder` [p.22] .

The `DOMBuilder` [p.22] will then allow the application to intercept any external entities (including the external DTD subset and external parameter entities) before including them.

Many DOM applications will not need to implement this interface, but it will be especially useful for applications that build XML documents from databases or other specialized input sources, or for applications that use URNs.

Note: `DOMEntityResolver` is based on the SAX2 [SAX] `EntityResolver` interface.

IDL Definition

```
interface DOMEntityResolver {
  DOMInputSource     resolveEntity(in DOMString publicId,
                                   in DOMString systemId,
                                   in DOMString baseURI);
};
```

Methods

`resolveEntity`
Allow the application to resolve external entities.
The `DOMBuilder` [p.22] will call this method before opening any external entity except the top-level document entity (including the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element); the application may request that the `DOMBuilder` resolve the entity itself, that it use an alternative URI, or that it use an entirely different input source.
Application writers can use this method to redirect external system identifiers to secure and/or local URIs, to look up public identifiers in a catalogue, or to read an entity from a database or other input source (including, for example, a dialog box).
If the system identifier is a URI, the `DOMBuilder` [p.22] must resolve it fully before reporting it to the application through this interface.
(***ED:*** See issue #4. An alternative would be to pass the URI out without resolving it, and to provide a base as an additional parameter. SAX resolves URIs first, and does not provide a base.)

Parameters

`publicId` of type `DOMString`
The public identifier of the external entity being referenced, or `null` if none was supplied.

`systemId` of type `DOMString`
The system identifier, a URI reference [IETF RFC 2396], of the external entity being referenced exactly as written in the source.
`baseURI` of type `DOMString`
The absolute base URI of the resource being parsed, or `null` if there is no base URI.

Return Value

`DOMInputSource` [p.15]	A `DOMInputSource` object describing the new input source, or `null` to request that the parser open a regular URI connection to the system identifier.

No Exceptions

Interface *DOMBuilderFilter*

`DOMBuilderFilters` provide applications the ability to examine nodes as they are being constructed during a parse. As each node is examined, it may be modified or removed, or the entire parse may be terminated early.

At the time any of the filter methods are called by the parser, the owner Document and DOMImplementation objects exist and are accessible. The document element is never passed to the `DOMBuilderFilter` methods, i.e. it is not possible to filter out the document element. The `Document`, `DocumentType`, `Notation`, and `Entity` nodes are not passed to the filter.

All validity checking while reading a document occurs on the source document as it appears on the input stream, not on the DOM document as it is built in memory. With filters, the document in memory may be a subset of the document on the stream, and its validity may have been affected by the filtering.

All default content, including default attributes, must be passed to the filter methods.

Any exception raised in the filter are ignored by the `DOMBuilder` [p.22] .

(***ED:*** The description of these methods is not complete)

IDL Definition

```
interface DOMBuilderFilter {

  // Constants returned by startElement and acceptNode
  const short               FILTER_ACCEPT                 = 1;
  const short               FILTER_REJECT                 = 2;
  const short               FILTER_SKIP                   = 3;
  const short               FILTER_INTERRUPT              = 4;

  unsigned short     startElement(in Element elt);
  unsigned short     acceptNode(in Node enode);
  readonly attribute unsigned long   whatToShow;
};
```

Definition group *Constants returned by startElement and acceptNode*

Constants returned by `startElement` and `acceptNode`.

Defined Constants

`FILTER_ACCEPT`
: Accept the node.

`FILTER_INTERRUPT`
: Interrupt the normal processing of the document.

`FILTER_REJECT`
: Reject the node abd its children.

`FILTER_SKIP`
: Skip this single node. The children of this node will still be considered.

Attributes

`whatToShow` of type `unsigned long`, readonly
: Tells the `DOMBuilder` [p.22] what types of nodes to show to the filter. See `NodeFilter` for definition of the constants. The constant `SHOW_ATTRIBUTE` is meaningless here, attribute nodes will never be passed to a `DOMBuilderFilter`.

Methods

`acceptNode`
: This method will be called by the parser at the completion of the parsing of each node. The node and all of its descendants will exist and be complete. The parent node will also exist, although it may be incomplete, i.e. it may have additional children that have not yet been parsed. Attribute nodes are never passed to this function.
  From within this method, the new node may be freely modified - children may be added or removed, text nodes modified, etc. The state of the rest of the document outside this node is not defined, and the affect of any attempt to navigate to, or to modify any other part of the document is undefined.
  For validating parsers, the checks are made on the original document, before any modification by the filter. No validity checks are made on any document modifications made by the filter.
  If this new node is rejected, the parser might reuse the new node or any of its descendants.

  Parameters

  `enode` of type `Node`
  : The newly constructed element. At the time this method is called, the element is complete - it has all of its children (and their children, recursively) and attributes, and is attached as a child to its parent.

  Return Value

`unsigned short`

- `FILTER_ACCEPT` if this `Node` should be included in the DOM document being built.
- `FILTER_REJECT` if the `Node` and all of its children should be rejected.
- `FILTER_SKIP` if the `Node` should be skipped and the `Node` should be replaced by all the children of the `Node`.
- `FILTER_INTERRUPT` if the filter wants to stop the processing of the document. Interrupting the processing of the document does no longer guarantee that the entire is XML *well-formed* [p.67] .

No Exceptions

`startElement`

This method will be called by the parser after each `Element` start tag has been scanned, but before the remainder of the `Element` is processed. The intent is to allow the element, including any children, to be efficiently skipped. Note that only element nodes are passed to the `startElement` function.

The element node passed to `startElement` for filtering will include all of the Element's attributes, but none of the children nodes. The Element may not yet be in place in the document being constructed (it may not have a parent node.)

A `startElement` filter function may access or change the attributes for the Element. Changing Namespace declarations will have no effect on namespace resolution by the parser.

For efficiency, the Element node passed to the filter may not be the same one as is actually placed in the tree if the node is accepted. And the actual node (node object identity) may be reused during the process of reading in and filtering a document.

Parameters

`elt` of type `Element`

The newly encountered element. At the time this method is called, the element is incomplete - it will have its attributes, but no children.

Return Value

`unsigned short`

- `FILTER_ACCEPT` if this `Element` should be included in the DOM document being built.
- `FILTER_REJECT` if the `Element` and all of its children should be rejected.
- `FILTER_SKIP` if the `Element` should be rejected. All of its children are inserted in place of the rejected `Element` node.
- `FILTER_INTERRUPT` if the filter wants to stop the processing of the document. Interrupting the processing of the document does no longer guarantee that the entire is XML *well-formed* [p.67] .

Returning any other values will result in unspecified behavior.

No Exceptions

Interface *DOMBuilder*

A interface to an object that is able to build a DOM tree from various input sources.

`DOMBuilder` provides an API for parsing XML documents and building the corresponding DOM document tree. A `DOMBuilder` instance is obtained from the `DOMImplementationLS` [p.10] interface by invoking its `createDOMBuilder`method.

As specified in [DOM Level 3 Core], when a document is first made available via the DOMBuilder:

- there is only one `Text` node for each block of text. The `Text` nodes are into "normal" form: only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent `Text` nodes nor empty `Text` nodes.
- it is expected that the `value` and `nodeValue` attributes of an `Attr` node initially return the *XML 1.0 normalized value*. However, if the features `validate-if-schema` and `datatype-normalization` are set to `true`, depending on the attribute normalization used, the attribute values may differ from the ones obtained by the XML 1.0 attribute normalization. If the feature `datatype-normalization` is not set to `true`, the XML 1.0 attribute normalization is guaranteed to occur, and if attributes list does not contain namespace declarations, the `attributes` attribute on `Element` node represents the property [attributes] defined in [XML Information set] .
  Issue Infoset:
  > XML Schemas does not modify the XML attribute normalization but represents their normalized value in an other information item property: [schema normalized value]
  > **Resolution:** XML Schema normalization only occurs if `datatype-normalization` is set to `true`.

Asynchronous `DOMBuilder` objects are expected to also implement the `events::EventTarget` interface so that event listeners can be registered on asynchronous `DOMBuilder` objects.

Events supported by asynchronous `DOMBuilder` are:

- **load**: The document that's being loaded is completely parsed, see the definition of `LSLoadEvent` [p.17]
- **progress**: Progress notification, see the definition of `LSProgressEvent` [p.17]

Note: All events defined in this specification use the namespace URI `"http://www.w3.org/2002/DOMLS"`.

`DOMBuilders` have a number of named features that can be queried or set. The name of `DOMBuilder` features must be valid XML names. Implementation specific features (extensions) should choose a implementation specific prefix to avoid name collisions.

Even if all features must be recognized by all implementations, being able to set a state (`true` or `false`) is not always required. The following list of recognized features indicates the definitions of each feature state, if setting the state to `true` or `false` must be supported or is optional and, which state is the default one:

`"canonical-form"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"cdata-sections"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"certified"`
: **`true`**
  : [*optional*]
  Assume, when XML 1.1 is supported, that the input is certified (see section 2.13 in [XML 1.1]).
: **`false`**
  : [*required*] (*default*)
  Don't assume that the input is certified (see section 2.13 in [XML 1.1]).

`"charset-overrides-xml-encoding"`
: **`true`**
  : [*required*] (*default*)
  If a higher level protocol such as HTTP [IETF RFC 2616] provides an indication of the character encoding of the input stream being processed, that will override any encoding specified in the XML declaration or the Text declaration (see also [XML 1.0] 4.3.3 "Character Encoding in Entities"). Explicitly setting an encoding in the `DOMInputSource` [p.15] overrides encodings from the protocol.
: **`false`**
  : [*required*]
  Any character set encoding information from higher level protocols is ignored by the parser.

`"comments"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"datatype-normalization"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"entities"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"infoset"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core]. Setting this feature to `true` will also force the feature `namespaces` to `true`.

`"namespaces"`

true
[*required*] (*default*)
Perform the namespace processing as defined in [XML Namespaces].
false
[*optional*]
Do not perform the namespace processing.

"namespace-declarations"
This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

"supported-mediatypes-only"
true
[*optional*]
Check that the media type of the parsed resource is a supported media type and call the error handler if an unsupported media type is encountered. The media types defined in [IETF RFC 3023] must be accepted.
false
[*required*] (*default*)
Don't check the media type, accept any type of data.

"unknown-characters"
true
[*required*] (*default*)
If, while verifying full normalization when XML 1.1 is supported, a processor encounters characters for which it cannot determine the normalization properties, then the processor will ignore any possible denormalizations caused by these characters.
false
[*optional*]
Report an fatal error if a character is encountered for which the processor can not determine the normalization properties.

"validate"
This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

"validate-if-schema"
This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

"whitespace-in-element-content"
This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

IDL Definition

```
interface DOMBuilder {
           attribute DOMEntityResolver entityResolver;
           attribute DOMErrorHandler errorHandler;
           attribute DOMBuilderFilter filter;
  void               setFeature(in DOMString name,
                                in boolean state)
                                        raises(DOMException);
  boolean            canSetFeature(in DOMString name,
                                   in boolean state);
```

```
  boolean            getFeature(in DOMString name)
                                    raises(DOMException);
  Document           parseURI(in DOMString uri);
  Document           parse(in DOMInputSource is);

  // ACTION_TYPES
  const unsigned short      ACTION_REPLACE                = 1;
  const unsigned short      ACTION_APPEND_AS_CHILDREN     = 2;
  const unsigned short      ACTION_INSERT_AFTER           = 3;
  const unsigned short      ACTION_INSERT_BEFORE          = 4;

  void               parseWithContext(in DOMInputSource is,
                                      in Node cnode,
                                      in unsigned short action)
                                        raises(DOMException);
};
```

Definition group *ACTION_TYPES*

A set of possible actions for the `parseWithContext` method.

Defined Constants

`ACTION_APPEND_AS_CHILDREN`
: Append the result of the input source as children of the context node. For this action to work, the context node must be an `Element` or a `DocumentFragment`.

`ACTION_INSERT_AFTER`
: Insert the result of parsing the input source after the context node. For this action to work the context nodes parent must be an `Element`.

`ACTION_INSERT_BEFORE`
: Insert the result of parsing the input source before the context node. For this action to work the context nodes parent must be an `Element`.

`ACTION_REPLACE`
: Replace the context node with the result of parsing the input source. For this action to work the context node must have a parent and the context node must be an `Element`, `Text`, `CDATASection`, `Comment`, `ProcessingInstruction`, or `EntityReference` node.

Attributes

`entityResolver` of type `DOMEntityResolver` [p.18]
: If a `DOMEntityResolver` [p.18] has been specified, each time a reference to an external entity is encountered the `DOMBuilder` will pass the public and system IDs to the entity resolver, which can then specify the actual source of the entity.
If this attribute is `null`, the resolution of entities in the document is implementation dependent.

`errorHandler` of type `DOMErrorHandler`
: In the event that an error is encountered in the XML document being parsed, the `DOMDocumentBuilder` will call back to the `errorHandler` with the error information. When the document loading process calls the error handler the node closest to where the error occured is passed to the error handler, if the implementation is unable to pass the node where the error occures the document Node is passed to the error handler. In addition to passing the Node closest to to where the error occured, the implementation

should also pass any other valuable information to the error handler, such as file name, line number, and so on. Mutations to the document from within an error handler will result in implementation dependent behaviour.

`filter` of type `DOMBuilderFilter` [p.19]

When the application provides a filter, the parser will call out to the filter at the completion of the construction of each `Element` node. The filter implementation can choose to remove the element from the document being constructed (unless the element is the document element) or to terminate the parse early. If the document is being validated when it's loaded the validation happens before the filter is called.

Methods

`canSetFeature`

Query whether setting a feature to a specific value is supported.

The feature name has the same form as a DOM hasFeature string.

Parameters

`name` of type `DOMString`

The feature name, which is a DOM has-feature style string.

`state` of type `boolean`

The requested state of the feature (`true` or `false`).

Return Value

`boolean`	`true` if the feature could be successfully set to the specified value, or `false` if the feature is not recognized or the requested value is not supported. The value of the feature itself is not changed.

No Exceptions

`getFeature`

Look up the value of a feature.

The feature name has the same form as a DOM hasFeature string

Parameters

`name` of type `DOMString`

The feature name, which is a string with DOM has-feature syntax.

Return Value

`boolean`	The current state of the feature (`true` or `false`).

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised when the `DOMBuilder` does not recognize the feature name.

`parse`

Parse an XML document from a resource identified by a `DOMInputSource` [p.15] .

Parameters

`is` of type `DOMInputSource` [p.15]
The `DOMInputSource` from which the source document is to be read.

Return Value

`Document`	If the `DOMBuilder` is a synchronous `DOMBuilder` the newly created and populated `Document` is returned. If the `DOMBuilder` is asynchronous then `null` is returned since the document object is not yet parsed when this method returns.

No Exceptions

`parseURI`

Parse an XML document from a location identified by a URI reference [IETF RFC 2396]. If the URI contains a fragment identifier (see section 4.1 in [IETF RFC 2396]), the behavior is not defined by this specification, but future versions of this specification might define the behavior.

Parameters

`uri` of type `DOMString`
The location of the XML document to be read.

Return Value

`Document`	If the `DOMBuilder` is a synchronous `DOMBuilder` the newly created and populated `Document` is returned. If the `DOMBuilder` is asynchronous then `null` is returned since the document object is not yet parsed when this method returns.

No Exceptions

`parseWithContext`

Parse an XML fragment from a resource identified by a `DOMInputSource` [p.15] and insert the content into an existing document at the position specified with the `contextNode` and `action` arguments. When parsing the input stream the context node is used for resolving unbound namespace prefixes. The `Document` node, attached to the context node, is used to resolved default attributes and entity references.
As the new data is inserted into the document at least one mutation event is fired per immediate child (or sibling) of context node.
If an error occurs while parsing, the caller is notified through the error handler.

Parameters

`is` of type `DOMInputSource` [p.15]
The `DOMInputSource` from which the source document is to be read. The source document must be an XML fragment, i.e. anything except an XML Document, a DOCTYPE, entities declarations, notations declarations, or XML or text declarations.

`cnode` of type `Node`
The node that is used as the context for the data that is being parsed. This node must be a `Document` node, a `DocumentFragment` node, or a node of a type that is allowed as a child of an element, e.g. it can not be an attribute node.

`action` of type `unsigned short`
: This parameter describes which action should be taken between the new set of node being inserted and the existing children of the context node. The set of possible actions is defined above.

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised when the `DOMBuilder` doesn't support this method. NO_MODIFICATION_ALLOWED_ERR: Raised if the context node is readonly.

No Return Value

`setFeature`

Set the state of a feature.

The feature name has the same form as a DOM hasFeature string.

It is possible for a `DOMBuilder` to recognize a feature name but to be unable to set its value.

Parameters

`name` of type `DOMString`
: The feature name.

`state` of type `boolean`
: The requested state of the feature (`true` or `false`).

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised when the `DOMBuilder` recognizes the feature name but cannot set the requested value. NOT_FOUND_ERR: Raised when the `DOMBuilder` does not recognize the feature name.

No Return Value

1.4. Save Interfaces

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "LS-Save" and "3.0" (respectively) to determine whether or not these interfaces are supported by the implementation. In order to fully support them, an implementation must also support the "Core" feature defined in the DOM Level 3 Core specification [DOM Level 3 Core]. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core].

Interface *DOMWriter*

`DOMWriter` provides an API for serializing (writing) a DOM document out in an XML document. The XML data is written to an output stream, the type of which depends on the specific language bindings in use.

During serialization of XML data, namespace fixup is done when possible as defined in [DOM Level 3 Core], Appendix B. [DOM Level 2 Core] allows empty strings as a real namespace URI. If the `namespaceURI` of a `Node` is empty string, the serialization will treat them as `null`, ignoring the prefix if any.

Note: should the remark on DOM Level 2 namespace URI included in the namespace algorithm in Core instead?

`DOMWriter` accepts any node type for serialization. For nodes of type `Document` or `Entity`, well formed XML will be created if possible. The serialized output for these node types is either as a Document or an External Entity, respectively, and is acceptable input for an XML parser. For all other types of nodes the serialized form is not specified, but should be something useful to a human for debugging or diagnostic purposes. Note: rigorously designing an external (source) form for stand-alone node types that don't already have one defined in [XML 1.0] seems a bit much to take on here.

Within a Document, DocumentFragment, or Entity being serialized, Nodes are processed as follows

- Documents are written including an XML declaration and a DTD subset, if one exists in the DOM. Writing a document node serializes the entire document.
- Entity nodes, when written directly by `writeNode` defined in the `DOMWriter` interface, output the entity expansion but no namespace fixup is done. The resulting output will be valid as an external entity.
- Entity reference nodes are serialized as an entity reference of the form "`&entityName;`" in the output. Child nodes (the expansion) of the entity reference are ignored.
- CDATA sections containing content characters that can not be represented in the specified output encoding are handled according to the "split-cdata-sections" feature.
  If the feature is `true`, CDATA sections are split, and the unrepresentable characters are serialized as numeric character references in ordinary content. The exact position and number of splits is not specified.
  If the feature is `false`, unrepresentable characters in a CDATA section are reported as errors. The error is not recoverable - there is no mechanism for supplying alternative characters and continuing with the serialization.
- DocumentFragment nodes are serialized by serializing the children of the document fragment in the order they appear in the document fragment.
- All other node types (Element, Text, etc.) are serialized to their corresponding XML source form.

Note: The serialization of a DOM Node does not always generate a *well-formed* [p.67] XML document, i.e. a `DOMBuilder` [p.22] might through fatal errors when parsing the resulting serialization.

Within the character data of a document (outside of markup), any characters that cannot be represented directly are replaced with character references. Occurrences of '<' and '&' are replaced by the predefined entities < and &. The other predefined entities (>, ', etc.) are not used; these characters can be included directly. Any character that can not be represented directly in the output character encoding is serialized as a numeric character reference.

Attributes not containing quotes are serialized in quotes. Attributes containing quotes but no apostrophes are serialized in apostrophes (single quotes). Attributes containing both forms of quotes are serialized in quotes, with quotes within the value represented by the predefined entity ". Any character that can not be represented directly in the output character encoding is serialized as a numeric character reference.

Within markup, but outside of attributes, any occurrence of a character that cannot be represented in the output character encoding is reported as an error. An example would be serializing the element <LaCañada/> with the encoding="us-ascii".

When requested by setting the `normalize-characters` feature on `DOMWriter`, all data to be serialized, both markup and character data, is W3C Text normalized according to the rules defined in [CharModel]. The W3C Text normalization process affects only the data as it is being written; it does not alter the DOM's view of the document after serialization has completed.

Namespaces are fixed up during serialization, the serialization process will verify that namespace declarations, namespace prefixes and the namespace URIs associated with Elements and Attributes are consistent. If inconsistencies are found, the serialized form of the document will be altered to remove them. The algorithm used for doing the namespace fixup while serializing a document is a combination of the algorithms used for lookupNamespaceURI and lookupNamespacePrefix .

(***ED:*** previous paragraph to be defined closer here.)

Any changes made affect only the namespace prefixes and declarations appearing in the serialized data. The DOM's view of the document is not altered by the serialization operation, and does not reflect any changes made to namespace declarations or prefixes in the serialized output.

While serializing a document the serializer will write out non-specified values (such as attributes whose `specified` is `false`) if the `discard-default-content` feature is set to `true`. If the `discard-default-content` flag is set to `false` and a schema is used for validation, the schema will be also used to determine if a value is specified or not. If no schema is used, the `specified` flag on attribute nodes is used to determine if attribute values should be written out.

Ref to Core spec (1.1.9, XML namespaces, 5th paragraph) entity ref description about warning about unbound entity refs. Entity refs are always serialized as `&foo;`, also mention this in the load part of this spec.

`DOMWriters` have a number of named features that can be queried or set. The name of `DOMWriter` features must be valid XML names. Implementation specific features (extensions) should choose an implementation dependent prefix to avoid name collisions.

Here is a list of features that must be recognized by all implementations.

Note: Using these features does affect the `Node` being serialized, only its serialized form is affected.

`"discard-default-content"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"canonical-form"`
: **`true`**
  : [*optional*]
  This formatting writes the document according to the rules specified in [Canonical XML]. Setting this feature to true will set the feature **`"format-pretty-print"`** to false.
: **`false`**
  : [*required*] (*default*)
  Do not canonicalize the output.

`"entities"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"format-pretty-print"`
: **`true`**
  : [*optional*]
  Formatting the output by adding whitespace to produce a pretty-printed, indented, human-readable form. The exact form of the transformations is not specified by this specification. Setting this feature to true will set the feature "canonical-form" to false.
: **`false`**
  : [*required*] (*default*)
  Don’t pretty-print the result.

`"namespaces"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"normalize-characters"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core]. Unlike in the Core, the default value for this feature is `true`.

`"split-cdata-sections"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"validate"`
: This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

`"unknown-characters"`
: **`true`**
  : [*required*] (*default*)
  If, while verifying full normalization when XML 1.1 is supported, a character is encountered for which the normalization properties cannot be determined, then ignore any possible denormalizations caused by these characters.

`false`
[*optional*]
Report an fatal error if a character is encountered for which the processor can not determine the normalization properties.

`"whitespace-in-element-content"`
This feature is equivalent to the one provided on `Document.setNormalizationFeature` in [DOM Level 3 Core].

IDL Definition

```
interface DOMWriter {
  void               setFeature(in DOMString name,
                                in boolean state)
                                        raises(DOMException);
  boolean            canSetFeature(in DOMString name,
                                   in boolean state);
  boolean            getFeature(in DOMString name)
                                        raises(DOMException);
           attribute DOMString       encoding;
           attribute DOMString       newLine;
           attribute DOMWriterFilter filter;
           attribute DOMErrorHandler errorHandler;
  boolean            writeNode(in DOMOutputStream destination,
                               in Node wnode);
  DOMString          writeToString(in Node wnode)
                                        raises(DOMException);
};
```

Attributes

`encoding` of type `DOMString`
The character encoding in which the output will be written.
The encoding to use when writing is determined as follows:
- If the encoding attribute has been set, that value will be used.
- If the encoding attribute is `null` or empty, but the item to be written, or the owner document of the item, specifies an encoding (i.e. the "actualEncoding" from the document) specified encoding, that value will be used.
- If neither of the above provides an encoding name, a default encoding of "UTF-8" will be used.

The default value is `null`.

`errorHandler` of type `DOMErrorHandler`
The error handler that will receive error notifications during serialization. The node where the error occured is passed to this error handler, any modification to nodes from within an error callback should be avoided since this will result in undefined, implementation dependent behavior.

`filter` of type `DOMWriterFilter` [p.35]
When the application provides a filter, the serializer will call out to the filter before serializing each Node. Attribute nodes are never passed to the filter. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.

`newLine` of type `DOMString`

The end-of-line sequence of characters to be used in the XML being written out. Any string is supported, but these are the recommended end-of-line sequences (using other character sequences than these recommended ones can result in a document that is either not serializable or not well-formed):

`null`

Use a default end-of-line sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by "End-of-Line Handling" ([XML 1.0], section 2.11) if the serialized content is XML 1.0 or "End-of-Line Handling" ([XML 1.1], section 2.11) if the serialized content is XML 1.1.

CR

The carriage-return character (#xD).

CR-LF

The carriage-return and line-feed characters (#xD #xA).

LF

The line-feed character (#xA).

The default value for this attribute is `null`.

Methods

`canSetFeature`

Query whether setting a feature to a specific value is supported.

The feature name has the same form as a DOM hasFeature string.

Parameters

`name` of type `DOMString`

The feature name, which is a DOM has-feature style string.

`state` of type `boolean`

The requested state of the feature (`true` or `false`).

Return Value

`boolean`	`true` if the feature could be successfully set to the specified value, or `false` if the feature is not recognized or the requested value is not supported. The value of the feature itself is not changed.

No Exceptions

`getFeature`

Look up the value of a feature.

The feature name has the same form as a DOM hasFeature string

Parameters

`name` of type `DOMString`

The feature name, which is a string with DOM has-feature syntax.

Return Value

`boolean`	The current state of the feature (`true` or `false`).

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised when the `DOMWriter` does not recognize the feature name.

`setFeature`

Set the state of a feature.

The feature name has the same form as a DOM hasFeature string.

It is possible for a `DOMWriter` to recognize a feature name but to be unable to set its value.

Parameters

`name` of type `DOMString`

The feature name.

`state` of type `boolean`

The requested state of the feature (`true` or `false`).

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised when the `DOMWriter` recognizes the feature name but cannot set the requested value. Raise a NOT_FOUND_ERR When the `DOMWriter` does not recognize the feature name.

No Return Value

`writeNode`

Write out the specified node as described above in the description of `DOMWriter`. Writing a Document or Entity node produces a serialized form that is well formed XML, when possible (Entity nodes might not always be well formed XML in themselves). Writing other node types produces a fragment of text in a form that is not fully defined by this document, but that should be useful to a human for debugging or diagnostic purposes.

If the specified encoding is not supported the error handler is called and the serialization is interrupted.

Parameters

`destination` of type `DOMOutputStream` [p.10]

The destination for the data to be written.

`wnode` of type `Node`

The `Document` or `Entity` node to be written. For other node types, something sensible should be written, but the exact serialized form is not specified.

Return Value

`boolean`	Returns `true` if `node` was successfully serialized and `false` in case a failure occured and the failure wasn't canceled by the error handler.

No Exceptions

`writeToString`

Serialize the specified node as described above in the description of `DOMWriter`. The result of serializing the node is returned as a DOMString (this method completely ignores all the encoding information available). Writing a Document or Entity node produces a serialized form that is well formed XML. Writing other node types produces a fragment of text in a form that is not fully defined by this document, but that should be useful to a human for debugging or diagnostic purposes.
Error handler is called if encoding not supported...

Parameters

`wnode` of type `Node`

The node to be written.

Return Value

`DOMString`	Returns the serialized data, or `null` in case a failure occured and the failure wasn't canceled by the error handler.

Exceptions

`DOMException`	DOMSTRING_SIZE_ERR: Raised if the resulting string is too long to fit in a `DOMString`.

Interface *DOMWriterFilter*

`DOMWriterFilters` provide applications the ability to examine nodes as they are being serialized. `DOMWriterFilter` lets the application decide what nodes should be serialized or not.

The `Document`, `DocumentType`, `Notation`, and `Entity` nodes are not passed to the filter.

IDL Definition

```
interface DOMWriterFilter : traversal::NodeFilter {
  readonly attribute unsigned long   whatToShow;
};
```

Attributes

`whatToShow` of type `unsigned long`, readonly

Tells the `DOMWriter` [p.28] what types of nodes to show to the filter. See `NodeFilter` for definition of the constants. The constants `SHOW_ATTRIBUTE`, `SHOW_DOCUMENT`, `SHOW_DOCUMENT_TYPE`, `SHOW_NOTATION`, and `SHOW_DOCUMENT_FRAGMENT` are meaningless here, those nodes will never be passed to a `DOMWriterFilter`. `Entity` nodes are not passed to the filter.

1.5. Issue List

1.5.1. Open Issues

Issue LS-Issue-90:

The interaction and relationships between all the DOMBuilder and DOMWriter features need to be defined, i.e. setting x will set y and unset z.

Issue LS-Issue-15:

System Exceptions. Loading involves file opens and reads, and these can result in a variety of system errors that may already have associated system exceptions. Should these system exceptions pass through as is, or should they be some how wrapped in DOMExceptions, or should there be a parallel set DOM Exceptions, or what?

Issue LS-Issue-21:

Define exceptions. A `DOMSystemException` needs to be defined as part of the error handling module that is to be shared with AS. Common I/O type errors need to be defined for it, so that they can be reported in a uniform way. A way to embed errors or exceptions from the OS or language environment is needed, to provide full information to applications that want it.

Issue LS-Issue-106:

There is no way to query the DOMImplementation for a specific DOMBuilder. support XML 1.1? support validation? namespaces=false?

1.5.2. Resolved Issues

Issue LS-Issue-1:

Should these methods be in a new interface, or should they be added to the existing DOMImplementation Interface? I think that adding them to the existing interface is cleaner, because it helps avoid an explosion of new interfaces.

The methods are in a separate interface in this description for convenience in preparing the doc, so that I don't need to edit Core to add the methods. (The same argument could perhaps be made for implementations.)

Resolution: The methods are in a separate DOMImplementationLS interface. Because Load/Save is an optional module, we don't want to add its to the core DOMImplementation interface.

Issue LS-Issue-2:

SAX handles the setting of parser attributes differently. Rather than having distinct getters and setters for each attribute, it has a generic setter and getter of named properties, where properties are specified by a URI. This has an advantage in that implementations do not need to extend the interface when providing additional attributes.

If we choose to use strings, their syntax needs to be chosen. URIs would make sense, except for the fact that these are just names that do not refer to any resources. Dereferencing them would be meaningless. Yet the direction of the W3C is that all URIs must be dereferencable, and refer to something on the web.

Resolution: Use strings for properties. Use Java package name syntax for the identifying names. The question was revisited at the July f2f, with the same conclusion. But some discussion of using URIs continues.

This issue was revisited once again at the 9/2000 meeting. Now all DOM properties or features will be short, descriptive names, and we will recommend that all vendor-specific extensions be prefixed

to avoid collisions, but will not make specific recommendations for the syntax of the prefix.

Issue LS-Issue-3:

It's not obvious what name to choose for the parser interface. Taking any of the names already in use by parser implementations would create problems when trying to support both the new API and the existing old API. That leaves out `DocumentBuilder` (Sun) and `DOMParser` (Xerces).

Resolution: This is issue really just a comment. The "resolution" is in the names appearing in the API.

Issue LS-Issue-4:

Question: should ResolveEntity pass a baseURI string back to the application, in addition to the publicId, systemId, and/or stream? Particularly in the case of an input stream.

Resolution: No. Sax2 explicitly says that the system ID URI must be fully resolved before passing it out to the entity resolve. We will follow SAX's lead on this unless some additional use case surfaces. This is from the 9/2000 f2f, and reverses an earlier decision.

2002-02-22: a baseURI parameter was added.

Issue LS-Issue-5:

When parsing a document that contains errors, should the whole document be decreed unusable, or should we say that portions prior to the point where the error was detected are OK?

Resolution: In the case of errors in the XML source, what, if any, document is returned is implementation dependent.

Issue LS-Issue-6:

The relationship between SAXExceptions and DOM exceptions seems confusing.

Resolution: This issue goes away because we are no longer using SAX. Any exceptions will be DOM Exceptions.

Issue LS-Issue-7:

Question: In the original Java definition, are the strings returned from the methods `SAXException.toString` and `SAXException.getMessage` always the same? If not, we need to add another attribute.

Resolution: No longer an issue because we are no longer using SAX.

Issue LS-Issue-8:

JAXP defines a mechanism, based on Java system properties, by which the Document Builder Factory locates the specific parser implementation to be used. This ability to redirect to different parsers is a key feature of JAXP. How this redirection works in the context of this design may be something that needs to be defined separately for each language binding.

This question was discussed at the July f2f, without resolution. Agreed that the feature is not critical to the rest of the API, and can be postponed.

Resolution: The issue is moving to core, where it is part of the bigger question of where does the DOM implementation come from, and how do multiple implementations coexist. Allowing separate, or mix-and-match, specification of the parser and the rest of the DOM is not generally practical because parsers generally have some degree of private knowledge about their DOMs.

Issue LS-Issue-9:

The use of interfaces from SAX2 raises some questions. The Java bindings for these interfaces need to be exactly the SAX2 definitions, including the original org.xml.sax package name.

The IDL presented here for these interfaces is an attempt to map the Java into IDL, but it will certainly not round-trip accurately - Java bindings generated from the IDL will not match the original Java.

The reasons for using the SAX interfaces are that they are well designed, widely implemented and

used, and provide what is needed. Designing something new would create confusion for application developers (which should be used?) and make extra work for implementers of the DOM, most of whom probably already provide SAX, all for no real gain.

Resolution: Problem is gone. We are not using SAX2. The design will borrow features and concepts from SAX2 when it makes sense to do so.

Issue LS-Issue-10:

Error Reporting. Loading will be reporting well-formedness and validation errors, just like AS. A common error reporting mechanism needs to be developed.

Resolution: Resolved, see errors.html

Issue LS-Issue-11:

Another Error Reporting Question. We decided at the June f2f that validity errors should not be exceptions. This means that a document load operation could encounter multiple errors. Should these be collected and delivered as some sort of collection at the (otherwise) successful completion of the load, or should there be some sort of callback? Callbacks are harder for applications to deal with.

Resolution: Provide a callback mechanism. Provide a default error handler that throws an exception and stops further processing. From July f2f.

Issue LS-Issue-12:

Definition of "Non-validating". Exactly how much processing is done by "non-validating" parsers is not fully defined by the XML specification. In particular, they are not required to read any external entities, but are not prohibited from doing so.

Another common user request: a mode that completely ignores DTDs, both and external. Such a parser would not conform to XML 1.0, however.

For the documents produced by a non-validating load to be the same, we need to tie down exactly what processing must be done. The XML Core WG also has question as an open issue .

Some discussion is at http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000JanMar/0192.html

Here is proposal: Have three classes of parsers

- Minimal. No external entities of any type are accessed. DTD subset is processes normally, as required by XML 1.0, including all entity definitions it contains.
- Non-Validating. All external entities are read. Does everything except validation.
- Validating. As defined by XML 1.0 rec.

Resolution: Use the options from SAX2. These provide separate flags for validation, reading of external general entities and reading of external parameter entities.

Issue LS-Issue-13:

Use of System or Language specific types for Input and Output

Loading and Saving requires that one of the possible sources or destinations of the XML data be some sort of stream that can be used with io streams or memory buffers, or anything else that might take or supply data. The type will vary, depending on the language binding.

The question is, what should be put into the IDL interfaces for these? Should we define an XML stream to abstract out the dependency, or use system classes directly in the bindings?

Resolution: Define IDL types for use in the rest of the interface definitions. These types will be mapped directly to system types for each language binding

Issue LS-Issue-14:

Should there be separate DOM modules for browser or scripting style loading (document.load("whatever")) and server style parsers? It's probably easy for the server style parsers to implement the browser style interface, but the reverse may not be `true`.

Resolution: Yes. A client application style API will be provided.

Issue LS-Issue-16:

Loading and saving of abstract schema's - DTDs or Schemas - outside of the context of a document is not addressed.

Resolution: See the `DOMASBuilder` interface in the AS spec

Issue LS-Issue-17:

Loading while validating using an already loaded abstract schema is not addressed. Applications should be able to load a abstract schema (issue 16), and then repeatedly reuse it during the loading of additional documents.

Resolution: See the `DOMASBuilder` interface in the AS spec

Issue LS-Issue-18:

For the list of parser properties, which must all implementations recognize, which settings must all implementations support, and which are optional?

Resolution: Done

Issue LS-Issue-19:

DOMOutputStream: should this be an interface with methods, or just an opaque type that maps onto an appropriate binding-specific stream type?

If we specify an actual interface with methods, applications can implement it to wrap any arbitrary destination that they may have. If we go with the system type it's simpler to output to that type of stream, but harder otherwise.

Resolution: Opaque.

Issue LS-Issue-20:

Action from September f2f to "add issues raised by schema discussion. What were these?

Resolution: nobody seems to remember this, no action taken

Issue LS-Issue-22:

What do the bindings for things like InputStream look like in ECMA Script? Tentative resolution - InputStream will map to a binding dependent class or interface. For environments where nothing appropriate exists, a new interface will be created. This question is still being discussed.

Resolution: will be left to the binding

Issue LS-Issue-23:

To Do: Add a method or methods to DOMBuilder that will provide information about a parser feature - is the name recognized, which (boolean) values are supported - without throwing exceptions.

Resolution: Done. Added canSetFeature.

Issue LS-Issue-24:

Clearly identify which of the parser properties must be recognized, and which of their settings must be supported by all conforming implementations.

Resolution: Done. All must be recognized.

Issue LS-Issue-25:

How does the validation property work in SAX, and how should it work for us? The default value in SAX2 is "true". Non-validating parsers only support a value of `false`. Does this mean that the default depends on the parser, or that some sort of an error happens if a parse is attempted before resetting the property, or what?

The same question applies to the External Entities properties too.

Resolution: Make the default value for the validation property be `false`.

Issue LS-Issue-26:

Do we want to rename the "auto-validation" property to "validate-if-cm"? Proposed at f2f. Resolution

unclear.

Resolution: Changed the name to "validate-if-cm".

Issue LS-Issue-27:

How is validation during document loading handled when there are multiple possible abstract schemas associated with the document? How is one selected? The same question exists for documents in general, outside of the context of loading. Resolving the question for loading probably needs to wait until the more general question is understood.

Resolution: Always use the active external AS if any and the active internal AS if any. Whenever you want to validate during parsing with a different Internal/External model you have to activate this Abstract Schema first.

Issue LS-Issue-29:

Should all properties except namespaces default to `false`? Discussed at f2f. I'm not so sure now. Some of the properties have somewhat non-standard behavior when `false` - leaving out ER nodes or whitespace, for example - and support of `false` will probably not even be required.

Resolution: Not all properties should default to `false`. But validation should.

Issue LS-Issue-28:

To do: add new parser property "createEntityNodes". default is `true`. Illegal for it to be `false` and createEntityReferenceNodes to be `true`.

(***ED:*** Is this really what we want?)

Resolution: new feature added.

Issue LS-Issue-30:

Possible additional parser features - option to not create CDATA nodes, and to merge CDATA contents with adjacent TEXT nodes if they exist. Otherwise just create a TEXT node.

Option to omit Comments.

Resolution: new feature added.

Issue LS-Issue-31:

We now have an option for fixing up namespace declarations and prefixes on serialization. Should we specify how this is done, so that the documents from different implementations of serialization will use the same declarations and prefixes, or should we leave the details up to the implementation?

Resolution: The exact form of the namespace fixup is implementation dependent. The only requirement is that all elements and attributes end up with the correct namespace URI.

Issue LS-Issue-32:

Mimetypes. If the input being parsed is from http or something else that supplies types, and the type is something other than text/xml, should we parse it anyhow, or should we complain. Should there be an option?

Tentative resolution: always parse, never complain. Reasons: 1. This is what all parsers do now, and no one has ever complained, at least not that I'm aware of. 2. Applications must have a pretty good reason to suspect that they're getting xml or they wouldn't have invoked the parser. 3. All the test would do is to take something that might have worked (xml that is not known to the server) and turn it into an error. Non-xml is exceptionally unlikely to successfully parse (be well formed.)

Resolution: See the `supported-mediatypes-only` feature on `DOMBuilder` [p.22] .

Issue LS-Issue-33:

Unicode Character Normalization Problems. It turns out that for some code pages, normalizing a Unicode representation, translating to the code page, then translating back to Unicode can result in un-normalized Unicode. Mark Davis says that this can happen with Vietnamese and maybe with Hebrew.

This means that the suggested W3C model of normalization on serialization (early normalization) may not work, and that the receiver of the data may need to normalize it again, just in case.
Resolution: The scenario described is a quality-of-implementation issue. A transcoder converting from the one of the troublesome code pages to a Unicode representation should be responsible for re-normalizing the output.

Issue LS-Issue-34:

Features 2.1.4.1, 2 - XML Fragment Support. Should these be dropped?
Resolution: The DOM WG decided to drop support for XML fragment loading in the DOM Level 3 Load-Save module due to lack of time to define the behavior in all the edge cases, future versions of this spec might address this issue.

Issue LS-Issue-35:

XPath based document load filter. It would be plausible to have a partial (filtered) document load based on selecting the portion of the document to load with an XPath expression. This facility could be in addition to the node-by-node filtering currently specified. Or we could drop the existing filter. Implementing an XPath based selective load would require that there be an XPath processor present in addition to the parser itself.
Resolution: The DOM Level 3 spec will not define an interface for doing XPath/XPointer type filtering, implementations are free to implement XPath/XPointer based filters on top of a DOMBuilderFilter.

Issue LS-Issue-36:

MIME Type checking for DOMASBuilder.
What MIME Type checking needs to be done for parsing schemas
Resolution: see DOMBuilder, DOMASBuilder is an extend of DOMBuilder, this issue is solved within DOMBuilder

Issue LS-Issue-37:

Internal ASModel serialization for DOMWriter.
What if the internal ASModel is an XML Schema ASModel. Currently there is no ASModel type. Adding an Internal ASModel can be any kind of schema. Should serialization somehow check the internal ASModel ? What about the internal subset, is it discarded when the AS spec is implemented ?
Resolution: An internal ASModel can’t be a schema according to the AS spec. The internal subset is discarded when an Abstract Schema is active and the AS spec is implemented

Issue LS-Issue-38:

Attribute Normalization.
Add a property to "attributeNormalization" to DOMWriter to support or discard Attribute Normalization during serialization to. Setting attributeNormalization will serialize attributes with unexpanded entity references (if any) regardless their childnode(s). This means that if a user is changing the child nodes of an entity reference node within an attribute and attributeNormalization is set to `true` during serialization that these changes are discarded during serialization.
Resolution: The normalization will be driven by the validation options on DOMBuilder, if a document is validated it will also be normalized, if the document is not validated then no normalization will occure.

Issue LS-Issue-39:

Validation at serialization time. Should we have an option for validating while serializing, what about validation errors, should we allow serializing non-valid DOM’s?
Resolution: No. Validation at serialization time will not be supported by this specification.

Issue LS-Issue-40:

Is the description of the DOMWriter option expand-entity-references acceptable?

Resolution: Yes, the description is acceptable.

Issue LS-Issue-41:

Do we need filter support in DOMWriter too?

Resolution: Not until we have good usecases for needing filters when serializing a node.

Issue LS-Issue-42:

Should all attributes on DOMInputSource be readonly? The DOM implementation will be passed an object that implements this interface and there's no need for the DOM implementation to ever modify any of those values.

Resolution: Yes, the application is responsible for implementing this interface, the DOM implementation should never modify an input source.

Issue LS-Issue-43:

What's a DOMReader in non-Java languages? Does this really belong in these language neutral interfaces?

Resolution: The DOMReader type should be defined as "Object" in ECMAScript.

Issue LS-Issue-44:

What should the DOMWriter do if the doctype name doesn't match the name of the document element? This is a validity error, not a wellformedness error so should this just be a normal validity error when serializing?

Resolution: This is only a validity error, and since this spec doesn't support validation at serialization time this will be ignored. If an implementation were to support validation at serialization time the error handler should be called in this case.

Issue LS-Issue-45:

How should validation work if there's a reference to both a schema and a DTD, should the parser validate against both, or only one, if only one, how does one select which one?

Resolution: Add a validate-against-dtd option that forces validation against the DTD even if there are other schemas referenced in the document.

Issue LS-Issue-46:

Should supporting async/sync loading be optional?

Resolution: Yes.

Issue LS-Issue-47:

Default attribute handling in DOMWriter needs to be defined for Level 1 elements.

Resolution: If Attr.specified is set to `false` then the attribute must be a level 1 node in which case this information can safely be used.

(***ED:*** This resolution needs to be put in sync with our `Attr.specified` discussion.)

Issue LS-Issue-48:

DOMWriter::writeNode takes a Node as an argument, shouldn't this be a Document?

Resolution: It should also be possible to serialize elements, adding xmlns declarations on the element that is serialized. Entities get serialized w/o binding element namespaces. Text nodes should be serialized too, and document fragments, cdata section and attributes too and entity reference (&foo;) and comments.

Issue LS-Issue-49:

Datatype normalization? I.e. stripping whitespace around integers n' such.

Resolution: No, but add option to not normalize when validating, "datatype-normalization" added.

Issue LS-Issue-50:
Should 'external-parameter-entities' be replaced by an "load-external-dtds-n'-stuff" option?
Resolution: yes, done, "external-parameter-entities" added.

Issue LS-Issue-51:
DOMBuilder::canSetFeature and ::supportsFeature are redundant, no?
Resolution: Yes, supportsFeature removed.

Issue LS-Issue-52:
Is the API dependencies on the Events spec acceptable?
Resolution: We're only reusing events API's, we're not requiring people to implement the events spec so this shouldn't be a problem.

Issue LS-ISSUE-53:
Doesn't the feature "external-dtd-subset" conflict with the XML 1.0 specifications standalone="true"?
Resolution: No, the standalone "attribute" in XML 1.0 is only a hint, and thus implementations are not required to do anything with it that matters for a DOM builder.

Issue LS-Issue-54:
"canonical-form" needs a correct reference to the spec for canonical XML.

Issue LS-Issue-55:
How should default attributes be dealt with wrt DOMBuilderFilter?
Resolution: All default content must be passed to the filter.

Issue LS-Issue-56:
Should we make it possible to `SKIP` an element in `DOMBuilderFilter::acceptNode`?
Resolution: Yes, done.

Issue LS-Issue-57:
`namespaceURI` in core can be empty string, how should that be dealt with in DOM LS?
Resolution: [DOM Level 2 Core] allows empty strings as a real namespace URI. If the `namespaceURI` of a `Node` is empty string, the serialization will treat them as `null`, ignoring the prefix if any.

Issue LS-Issue-58:
Some features should not be required for parseWithContext() (such as validate, validate-if-schema, whitespace-in-element-content, external-dtd-subset, ...), what are these options, and how do we describe this?

Issue LS-Issue-59:
ACTION_APPEND is confusing, can we clarify it?
Resolution: make it ACTION_APPEND_AS_CHILDREN (2002-01-28)

Issue LS-Issue-60:
DOMEntityResolver::baseURI, should it be absolute or can it be relative?
Resolution: make it absolute. (2002-01-28)

Issue LS-Issue-61:
How to use an empty document with parseWithContext?
Resolution: As of today, it is not possible to have an empty Document using the DOM Core, so we don't consider this as an issue. However, following the discussion on having support for empty Document in the Core, this issue might be reopened.

Issue LS-Issue-62:
createDOMBuilder: If MODE_SYNCHRONOUS and MODE_ASYNCHRONOUS are the only anticipated values, then a boolean parameter would be preferred. If it stays a unsigned short, then

there needs to be a exception for unrecognized values.
Resolution: We keep the unsigned hsort for future possible extension.
"NOT_SUPPORTED_ERR: Raised if the requested mode is not supported."

Issue LS-Issue-63:
createDOMBuilder: The description of the return value mentions the type parameter, however the method has no parameters.
Resolution: Fixed.

Issue LS-Issue-64:
createDOMWriter: Being able to create an asynchronous writer would be desirable. I'd add a mode parameter to parallel createDOMBuilder.
Resolution: This will not be addressed by this version of the DOM LS spec.

Issue LS-Issue-65:
DOMBuilder.errorHandler: Passing "the node closest to where the error occurred" is really vague. Especially if the problem is a well-formedness or other fatal error. An character offset and/or text fragment would be more useful for error diagnosis. Passing null if the closest node could not be determined would be cleaner than passing the document.
Resolution: Description updated to indicate that any other available position information should also be passed to the error handler.

Issue LS-Issue-66:
parse and parseURI DOMBuilder methods: Returning null for asynchrous DOMBuilder's would make it difficult to express DocumentLS.load in terms of DOMBuilder.parse. Since DocumentLS appears to be a convenience interface, everything should be expressible in terms of the more general interfaces.
Resolution: DocumentLS.load and DOMBuilder.parse* are two completely different animals. One can most likely not be implemented in terms of using the other, and this will not change. DocumentLS.load is defined as it is for compatibility with existing implementations, and that won't be changed. Returning a document from an async parse method on the DOMBuilder is just not practical since you don't know at the time when the parse method returns what type of document you'll need. No change.

Issue LS-Issue-67:
DOMBuilder.parseURI: Specifying a behavior for URI's containing fragment identifier would seem desirable. I'd suggest ignoring the fragment identifier, but throwing an exception would be better than leaving it unspecified.
Resolution: Description updated, no exception, undefined behavior for now but future versions might define the behavior.

Issue LS-Issue-68:
DOMBuilder.parseWithContext: Should throw DOMSystemExceptions. Should throw NO_MODIFICATION_ALLOWED_ERR if context node (or parent) is read-only. Returning the created node would be desirable.
Resolution: Exception added. But the created node can't be returned since there might be more than one node created.

Issue LS-Issue-69:
How does DOMBuilder.parseWithContext interact with any event listeners registered on the context node or its ancestors?
Resolution: Description on what mutation events are fired when using parseWithContext() added.

Issue LS-Issue-70:

DOMBuilder.setFeature: Several features force other features to specific values, but there is no defined behavior if you try to override the forced value, for example, setting external-parameter-entities to false after setting validation to true. I would suggest throwing an exception.

Resolution: No exceptions will be thrown. See issue 90.

Issue LS-Issue-71:

DOMWriter.encoding attribute: The second bullet should describe how Document.encoding or Document.actualEncoding are used to determine the encoding.

Issue LS-Issue-72:

DOMWriter.encoding attribute: Should throw an exception on setting if the encoding in not supported.

Resolution: Definition of DOMWriter.writeNode() updated, no exception thrown on setting the encoding.

Issue LS-Issue-73:

DOMWriter.encoding attribute: There should be a list of required encodings (at minimum UTF-8 and UTF-16)

Resolution: No list will be defined in the DOM spec. The XML specification defines some required encodings, we won't define anything more than that.

Issue LS-Issue-74:

DOMWriter.lastEncoding attribute: I'd prefer a method where I'd pass in a Node and get the encoding that would be used. Don't like the statefulness of the attribute.

Resolution: The LS ET decided to remove this attribute completely since it doesn't really serve any valid purpose. The LS spec will not define an API for finding out what encoding would be used for a particular Node.

Issue LS-Issue-75:

DOMWriter.errorHandler: Might be more general than just errors, could be reporting progress or other details (such as the selected encoding) or participating in filtering.

Resolution: No, the error handler is an error handler and nothing more. Other API's should be defined for things like progress notifications or other such callbacks. Unless someone provides a compelling usecase for changing this, it won't change.

Issue LS-Issue-76:

DOMWriter.newLine: Should probably be a unsigned short with constants for the supported values like other enumerations in the spec.

Resolution: Description updated, this will remain a string and the definition was relaxed to support any string so that future unicode newlines n' such can be used w/o an API change.

Issue LS-Issue-77:

DOMWriter.setFeature method: Should have an defined exception for inconsistent features, like turning pretty-printing on after setting canonical-form to true.

Resolution: See issue 90.

Issue LS-Issue-78:

DOMWriter.writeNode method: Writing a Document or Entity node... well formed XML. Why would writing an entity node be well formed XML?

Resolution: Description updated.

Issue LS-Issue-79:

DOMWriter.writeToString method: How is this affected by encoding? It will be represented

internally as UTF-16 on most binding, but users who have set encoding to ISO-8859-1 or US-ASCII might expect no code points higher than 255 or 127 respectively so they can naively write out the string to a file later.

Resolution: writeToString() always writes into a DOMString, which means it's always UTF16. The encoding information available is always ignored in writeToString(). Description updated to reflect this.

Issue LS-Issue-80:

DOMInputSource Interface: I don't like the multiple personalities of this interface. Instead of creating a DOMInputSource and then customizing it by setting attributes, I'd prefer multiple create (createSourceFromURI, createSourceFromString, etc), methods on DOMImplementationLS and only the minimum read-only attributes on DOMInputSource.

Resolution: Won't change, there are too many combinations of input sources to define specific factory methods for all combinations.

Issue LS-Issue-81:

DOMEntityResolver Interface: "for applications that use URI types other than URIs" Did you mean URL's.

Resolution: Description updated.

Issue LS-Issue-82:

DOMBuilderFilter.acceptNode and .startContainer: If the return value was a Node, then a Filter could:

1. return the passed enode to have the element inserted.
2. return null to have the element rejected
3. return a DocumentFragment for SKIP

Resolution: Won't change, this would make it more complicated and more expensive to implement than with the current proposal.

Issue LS-Issue-83:

DOMBuilderFilter.acceptNode and .startContainer: substitute a replacement element created with Document.createElement[NS]

Resolution: No, such mutations to the tree from a filter is not allowed by this spec.

Issue LS-Issue-84:

DOMBuilderFilter.acceptNode and .startContainer: It should be possible to throw an exception in acceptNode and startContainer to stop the parse.

Terminating parsing from a DOMBuilderFilter: The description of the DOMBuilderFilter states that parsing can be terminated early using a filter, but doesn't give a specific recommendation or mechanism regarding how to do this. Should this be binding-specific, or is there a particular DOM exception which should be raised?

Resolution: Use `FILTER_INTERRUPT` if you want to stop the processing of the document. Interrupting the processing of the document does no longer guarantee that the entire is XML well-formed.

Issue LS-Issue-85:

DocumentLS interface: An isLoaded or ReadyState attribute would be strongly desirable to determine that an async document was loaded without registering an event listener.

Resolution: This has been discussed and proposed before, and so far all proposals have been turned down. The load listener can be used for being notified about when a document is done loading, that lets you do everything a ReadyState or isLoaded attribute would do for you, cleaner and more efficiently (i.e. no polling of state, or anything like that).

Issue LS-Issue-86:

DocumentLS.load: Should an exception be raised if you attempt to start a second async load when one is already in progress?

Resolution: No, no exception. Calling .load() while a load is in progress on that same document will cancel the current load and start the new one.

Issue LS-Issue-87:

Document.loadXML: How would any XML declaration specifying an encoding be handled.

Resolution:

Issue LS-Issue-88:

DOMErrorHandler Interface: Called functions should be able to throw some type of exception or return an object to stop the parse and raise an exception to the caller of parse. Those exceptions would need to be added to the list of potential exceptions on the parse calls.

Resolution: Error handler methods can not throw exceptions. The main reason for this is that in the async loading case there's none on the receiving end of the call to the error handler that would be able to deal with the exception. And besides, exceptions are for exceptional cases, this would not be such a case.

Issue LS-Issue-89:

The description of the whatToShow attribute in DOM3 Load and Save for both DOMWriterFilter and DOMBuilderFilter is unclear. For example, if I set whatToShow to NodeFilter.SHOW_ELEMENT does this mean that only element nodes will be output? or does it mean that only element nodes will be passed to the filter for further consideration while other kinds of nodes will be output without being checked through the filter?

Resolution: The description is already pretty clear on this, no change.

Issue LS-Issue-91:

DOMBuilder.entityResolver: The description should describe what support a builder is expected to provide if the resolver is not specified. When a new builder is created, should a default resolver be exposed via this attribute, to allow client code to "wrap" a basic resolver, or should the default value be null? (This kind of information would be helpful for many attributes in the DOM spec.)

Resolution: by default the parser is free to do whatever he wants regarding entities resolution.

Issue LS-Issue-92:

DOMBuilder.errorHandler: When a new builder is created, should a error handler be exposed via this attribute, to allow client code to "wrap" a handler, or should the default value be null?
if no error handler, then throw exceptions?

Resolution: DOMBuilder.errorHandler might expose a default error handler at creation time. (changed DOMImplementationLS.createDOMBuilder description).

Issue LS-Issue-93:

DOMBuilderFilter.whatToShow: The description of this attribute states that attribute nodes will never be passed to the filter, and the description of the filter interface also states that the document element will not be passed to the filter. What about the Document, DocumentType, Notation, and Entity nodes?

Resolution: Document, DocumentType, Notation and entities are not passed to the filter.

Issue LS-Issue-94:

DocumentLS.saveXML: Why would the return value ever be null?

Resolution: "or null in case an error occurred."

Issue LS-Issue-95:

The DOMBuilder supports a "feature" called "create-entity-nodes"; is there a reason to also define

"create-notation-nodes"? There's definitively less need to provide a filter of this sort. Perhaps there should be an option to not build the DocumentType node at all, even if present? "processing-instructions" ?

Resolution: Not enough use cases to include those features. Use DOMBuilderFilter. For DocumentType, implications on validation are not certain, so we don't plan to support it for the moment.

Issue LS-Issue-96:

The description of serializing character data and attributes is at variance with XML C14N rules; it seems preferable to stay consistent with C14N where possible, or at least to better motivate any departures.

For example, the description:

"Attributes containing quotes but no apostrophes are serialized in apostrophes (single quotes). Attributes containing both forms of quotes are serialized in quotes, with quotes within the value represented by the predefined entity "."

varies from C14N which never uses single quotes but always replaces a quotation mark in the attribute value with ".

Somebody should carefully review this text with respect to C14N rules, and either use C14N rules or provide feature options on DOMWriter that allows the user of DOMWriter to choose the appropriate serialization.

Resolution: use canonical-form for C14N, otherwise, we keep the way it is currently defined.

Issue LS-Issue-97:

Under the description of DOMWriter appears the following:

" When serializing a document the DOMWriter checks to see if the document element in the document is a DOM Level 1 element or a DOM Level 2 (or higher) element (this check is done by looking at the localName of the root element). If the root element is a DOM Level 1 element then the DOMWriter will issue an error if a DOM Level 2 (or higher) element is found while serializing. Likewise if the document element is a DOM Level 2 (or higher) element and the DOMWriter sees a DOM Level 1 element an error is issued. Mixing DOM Level 1 elements with DOM Level 2 (or higher) is not supported."

I'm not sure what this is saying. Is it describing a scenario where multiple implementations are simultaneously used with a single API and a document which was instantiated by a Level 1 implementation has an element which was instantiated by a Level 2 implementation? Wouldn't it be an error to import a Level 2 node into a Level 1 document in the first place? Or wouldn't such an import effectively downcast that Level 2 node to its Level 1 counterpart?

If, on the other hand, this language is not talking about multiple implementations, then how is it possible to have a Level 2 implementation create a Level 1 element? Any element created by a Level 2 implementation will be a Level 2 element.

Resolution: not an issue anymore (text removed).

Issue LS-Issue-98:

Regarding the "namespace-declarations" feature of DOMBuilder, which is defaulted as "true", meaning *"include the namespace declaration attributes, specified or defaulted from the schema or the DTD, in the DOM document"*, how does this correlate with the following statements:

1. in DOM-3 Core, under Element, it is stated *"The properties [namespace attributes] and [in-scope namespaces] defined in [XML Information set] are not accessible from DOM Level 3 Core."*; and
2. in DOM-3 LS, under 2.1.3, it is stated *"All information to be serialized should be available via*

the normal DOM API."

Unless I am missing something (which is probably the case), these latter two statements would seem to indicate that it is impossible to support "namespace-declarations" as presently defined.

Resolution: DOM 3 Core was fixed.

Issue LS-Issue-99:

DOMBuilder.parseWithContext: It states that the context node should be used for namespace resolution, does the same apply to default attributes and entity references, are these to be taken from the document on which the parse is done?

Resolution: default attributes and entity referecences are taken from the Document attached to the context node.

Issue LS-Issue-100:

Is document fragment going to be defined. Since you do not have to parse a complete document at that point, I suppose both

```
<foo/><bar/>
```

and

```
foobar
```

are valid fragments, but is there an exact definition for this? I am particularly interested whether a document type is allowed in the input source that is the argument of this method. Since the input may also be a document, I suppose the answer is 'yes', but I think that would require implementations (or maybe just mine?) to 'double parse' or at least examine the stream a little, as the fragments

```
 <?xml version="1.0"?>
 <!DOCTYPE foo>
 foo
and
 foobar
```

would have to be handled differently (one is wellformed xml, the other is not, and there is at least one parseWithContext-usage where an input with a doctype would lead to a wellformed result).

Resolution: parseWithContext can take an XML fragment: i.e. anything except an XML Document, a DOCTYPE, entities declarations, or notations declarations.

Issue LS-Issue-101:

reconsider your removal of the namespaces feature

Resolution: added back in the draft.

Issue LS-Issue-102:

DOMWriterFilter/DOMBuilderFilter: do you pass the document element, document type, document, etc. to the filter?

Resolution: Document, DocumentType, Notation and entities are not passed to the filter.

Issue LS-Issue-103:

Current proposal: parse raises DOMIOException. All DOMIOException must also be reported to the DOMErrorHandler.

do we need to report and throw the exception? report seems to be enough but if no error handler was set, doesn't seem right to die like that...

Resolution: No changes.

Issue LS-Issue-104:

the spec says DocumentLS "uses the default features". Does it mean that no validation, etc will be performed? basically XML 1.0 loading ? the document will be loading, attribute value normalization XML 1.0 will be performed, but no validation will occure (even if users previously set

setNormalizationFeature() on the document)?

Resolution: entities - default is false, which is different from Core.

Issue LS-Issue-105:

DocumentLS.saveXML isn't clear enough as to whether it's deep or not. If the snode parameter is null, the whole document is serialized. OK, that's deep. But if it is non-null, then ONLY the Node provided is serialized? What if one wishes to use DocumentLS.saveXML to serialize a specific node AND its children? I think it might be nice to have a boolean deep argument for this method, and specify that the provided Node, and all its children, may be serialized if the deep parameter is true. If false, then only the provided Node is serialized.

Resolution: DocumentLS.saveXML saves the document or the given node and all its decendants to a string.

Appendix A: IDL Definitions

This appendix contains the complete OMG IDL [OMG IDL] for the Level 3 Document Object Model Abstract Schemas and Load and Save definitions.

The IDL files are also available as: http://www.w3.org/TR/2002/WD-DOM-Level-3-LS-20020725/idl.zip

ls.idl:

```
// File: ls.idl

#ifndef _LS_IDL_
#define _LS_IDL_

#include "dom.idl"
#include "events.idl"
#include "traversal.idl"

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

  typedef   Object DOMInputStream;

  typedef   Object DOMOutputStream;

  typedef   Object DOMReader;

  typedef dom::DOMString DOMString;
  typedef dom::Node Node;
  typedef dom::Element Element;
  typedef dom::DOMErrorHandler DOMErrorHandler;
  typedef dom::Document Document;

  interface DOMBuilder;
  interface DOMWriter;
  interface DOMInputSource;
  interface DOMWriterFilter;

  interface DOMImplementationLS {

    // DOMIMplementationLSMode
    const unsigned short      MODE_SYNCHRONOUS              = 1;
    const unsigned short      MODE_ASYNCHRONOUS             = 2;

    DOMBuilder         createDOMBuilder(in unsigned short mode,
                                        in DOMString schemaType)
                                        raises(dom::DOMException);
    DOMWriter          createDOMWriter();
    DOMInputSource     createDOMInputSource();
  };

  interface DocumentLS {
             attribute boolean         async;
```

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

  void               abort();
  boolean            load(in DOMString uri);
  boolean            loadXML(in DOMString source);
  DOMString          saveXML(in Node snode)
                                raises(dom::DOMException);
};

interface DOMInputSource {
           attribute DOMInputStream  byteStream;
           attribute DOMReader       characterStream;
           attribute DOMString       stringData;
           attribute DOMString       encoding;
           attribute DOMString       publicId;
           attribute DOMString       systemId;
           attribute DOMString       baseURI;
};

interface DOMEntityResolver {
  DOMInputSource     resolveEntity(in DOMString publicId,
                                   in DOMString systemId,
                                   in DOMString baseURI);
};

interface DOMBuilderFilter {

  // Constants returned by startElement and acceptNode
  const short               FILTER_ACCEPT                 = 1;
  const short               FILTER_REJECT                 = 2;
  const short               FILTER_SKIP                   = 3;
  const short               FILTER_INTERRUPT              = 4;

  unsigned short     startElement(in Element elt);
  unsigned short     acceptNode(in Node enode);
  readonly attribute unsigned long   whatToShow;
};

interface DOMBuilder {
           attribute DOMEntityResolver entityResolver;
           attribute DOMErrorHandler errorHandler;
           attribute DOMBuilderFilter filter;
  void               setFeature(in DOMString name,
                                in boolean state)
                                      raises(dom::DOMException);
  boolean            canSetFeature(in DOMString name,
                                   in boolean state);
  boolean            getFeature(in DOMString name)
                                      raises(dom::DOMException);
  Document           parseURI(in DOMString uri);
  Document           parse(in DOMInputSource is);

  // ACTION_TYPES
  const unsigned short      ACTION_REPLACE                 = 1;
  const unsigned short      ACTION_APPEND_AS_CHILDREN      = 2;
  const unsigned short      ACTION_INSERT_AFTER            = 3;
  const unsigned short      ACTION_INSERT_BEFORE           = 4;
```

```
    void               parseWithContext(in DOMInputSource is,
                                        in Node cnode,
                                        in unsigned short action)
                                        raises(dom::DOMException);
  };

  interface DOMWriter {
    void               setFeature(in DOMString name,
                                  in boolean state)
                                        raises(dom::DOMException);
    boolean            canSetFeature(in DOMString name,
                                     in boolean state);
    boolean            getFeature(in DOMString name)
                                        raises(dom::DOMException);
             attribute DOMString       encoding;
             attribute DOMString       newLine;
             attribute DOMWriterFilter filter;
             attribute DOMErrorHandler errorHandler;
    boolean            writeNode(in DOMOutputStream destination,
                                 in Node wnode);
    DOMString          writeToString(in Node wnode)
                                        raises(dom::DOMException);
  };

  interface LSLoadEvent : events::Event {
    readonly attribute Document        newDocument;
    readonly attribute DOMInputSource  inputSource;
  };

  interface LSProgressEvent : events::Event {
    readonly attribute DOMInputSource  inputSource;
    readonly attribute unsigned long   position;
    readonly attribute unsigned long   totalSize;
  };

  interface DOMWriterFilter : traversal::NodeFilter {
    readonly attribute unsigned long   whatToShow;
  };
};

#endif // _LS_IDL_
```

ls.idl:

Appendix B: Java Language Binding

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

The Java files are also available as
http://www.w3.org/TR/2002/WD-DOM-Level-3-LS-20020725/java-binding.zip

org/w3c/dom/ls/DOMImplementationLS.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.DOMException;

public interface DOMImplementationLS {
    // DOMIMplementationLSMode
    public static final short MODE_SYNCHRONOUS          = 1;
    public static final short MODE_ASYNCHRONOUS         = 2;

    public DOMBuilder createDOMBuilder(short mode,
                                       String schemaType)
                                       throws DOMException;

    public DOMWriter createDOMWriter();

    public DOMInputSource createDOMInputSource();

}
```

org/w3c/dom/ls/DocumentLS.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.Node;
import org.w3c.dom.DOMException;

public interface DocumentLS {
    public boolean getAsync();
    public void setAsync(boolean async)
                                                throws DOMException;

    public void abort();

    public boolean load(String uri);

    public boolean loadXML(String source);

    public String saveXML(Node snode)
                          throws DOMException;

}
```

org/w3c/dom/ls/DOMInputSource.java:

```
package org.w3c.dom.ls;

public interface DOMInputSource {
    public java.io.InputStream getByteStream();
    public void setByteStream(java.io.InputStream byteStream);

    public java.io.Reader getCharacterStream();
    public void setCharacterStream(java.io.Reader characterStream);

    public String getStringData();
    public void setStringData(String stringData);

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

    public String getPublicId();
    public void setPublicId(String publicId);

    public String getSystemId();
    public void setSystemId(String systemId);

    public String getBaseURI();
    public void setBaseURI(String baseURI);

}
```

org/w3c/dom/ls/LSLoadEvent.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.Document;
import org.w3c.dom.events.Event;

public interface LSLoadEvent extends Event {
    public Document getNewDocument();

    public DOMInputSource getInputSource();

}
```

org/w3c/dom/ls/LSProgressEvent.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.events.Event;

public interface LSProgressEvent extends Event {
    public DOMInputSource getInputSource();

    public int getPosition();
```

```
    public int getTotalSize();

}
```

org/w3c/dom/ls/DOMEntityResolver.java:

```
package org.w3c.dom.ls;

public interface DOMEntityResolver {
    public DOMInputSource resolveEntity(String publicId,
                                        String systemId,
                                        String baseURI);

}
```

org/w3c/dom/ls/DOMBuilderFilter.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

public interface DOMBuilderFilter {
    // Constants returned by startElement and acceptNode
    public static final short FILTER_ACCEPT             = 1;
    public static final short FILTER_REJECT             = 2;
    public static final short FILTER_SKIP               = 3;
    public static final short FILTER_INTERRUPT          = 4;

    public short startElement(Element elt);

    public short acceptNode(Node enode);

    public int getWhatToShow();

}
```

org/w3c/dom/ls/DOMBuilder.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.DOMException;
import org.w3c.dom.DOMErrorHandler;

public interface DOMBuilder {
    public DOMEntityResolver getEntityResolver();
    public void setEntityResolver(DOMEntityResolver entityResolver);

    public DOMErrorHandler getErrorHandler();
    public void setErrorHandler(DOMErrorHandler errorHandler);

    public DOMBuilderFilter getFilter();
```

```
    public void setFilter(DOMBuilderFilter filter);

    public void setFeature(String name,
                           boolean state)
                           throws DOMException;

    public boolean canSetFeature(String name,
                                 boolean state);

    public boolean getFeature(String name)
                              throws DOMException;

    public Document parseURI(String uri);

    public Document parse(DOMInputSource is);

    // ACTION_TYPES
    public static final short ACTION_REPLACE            = 1;
    public static final short ACTION_APPEND_AS_CHILDREN = 2;
    public static final short ACTION_INSERT_AFTER       = 3;
    public static final short ACTION_INSERT_BEFORE      = 4;

    public void parseWithContext(DOMInputSource is,
                                 Node cnode,
                                 short action)
                                 throws DOMException;

}
```

org/w3c/dom/ls/DOMWriter.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.Node;
import org.w3c.dom.DOMException;
import org.w3c.dom.DOMErrorHandler;

public interface DOMWriter {
    public void setFeature(String name,
                           boolean state)
                           throws DOMException;

    public boolean canSetFeature(String name,
                                 boolean state);

    public boolean getFeature(String name)
                              throws DOMException;

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

    public String getNewLine();
    public void setNewLine(String newLine);

    public DOMWriterFilter getFilter();
    public void setFilter(DOMWriterFilter filter);
```

```
    public DOMErrorHandler getErrorHandler();
    public void setErrorHandler(DOMErrorHandler errorHandler);

    public boolean writeNode(java.io.OutputStream destination,
                             Node wnode);

    public String writeToString(Node wnode)
                                throws DOMException;

}
```

org/w3c/dom/ls/DOMWriterFilter.java:

```
package org.w3c.dom.ls;

import org.w3c.dom.traversal.NodeFilter;

public interface DOMWriterFilter extends NodeFilter {
    public int getWhatToShow();

}
```

org/w3c/dom/ls/DOMWriterFilter.java:

Appendix C: ECMAScript Language Binding

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

Properties of the **DOMImplementationLS** Constructor function:

DOMImplementationLS.MODE_SYNCHRONOUS

The value of the constant **DOMImplementationLS.MODE_SYNCHRONOUS** is **1**.

DOMImplementationLS.MODE_ASYNCHRONOUS

The value of the constant **DOMImplementationLS.MODE_ASYNCHRONOUS** is **2**.

Objects that implement the **DOMImplementationLS** interface:

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

createDOMBuilder(mode, schemaType)

This function returns an object that implements the **DOMBuilder** interface.
The **mode** parameter is a **Number**.
The **schemaType** parameter is a **String**.
This function can raise an object that implements the **DOMException** interface.

createDOMWriter()

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

createDOMInputSource()

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

Objects that implement the **DocumentLS** interface:

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

async

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

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

abort()

This function has no return value.

load(uri)

This function returns a **Boolean**.
The **uri** parameter is a **String**.

loadXML(source)

This function returns a **Boolean**.
The **source** parameter is a **String**.

saveXML(snode)

This function returns a **String**.
The **snode** parameter is an object that implements the **Node** interface.
This function can raise an object that implements the **DOMException** interface.

Objects that implement the **DOMInputSource** interface:

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

byteStream

This property is an object that implements the **Object** interface.

characterStream

This property is an object that implements the **this is an error and shouldn't be used.** interface.

stringData
This property is a **String**.
encoding
This property is a **String**.
publicId
This property is a **String**.
systemId
This property is a **String**.
baseURI
This property is a **String**.

Objects that implement the **LSLoadEvent** interface:

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

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

newDocument
This read-only property is an object that implements the **Document** interface.
inputSource
This read-only property is an object that implements the **DOMInputSource** interface.

Objects that implement the **LSProgressEvent** interface:

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

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

inputSource
This read-only property is an object that implements the **DOMInputSource** interface.
position
This read-only property is a **Number**.
totalSize
This read-only property is a **Number**.

Objects that implement the **DOMEntityResolver** interface:

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

resolveEntity(publicId, systemId, baseURI)
This function returns an object that implements the **DOMInputSource** interface.
The **publicId** parameter is a **String**.
The **systemId** parameter is a **String**.
The **baseURI** parameter is a **String**.

Properties of the **DOMBuilderFilter** Constructor function:

DOMBuilderFilter.FILTER_ACCEPT
The value of the constant **DOMBuilderFilter.FILTER_ACCEPT** is **1**.
DOMBuilderFilter.FILTER_REJECT
The value of the constant **DOMBuilderFilter.FILTER_REJECT** is **2**.
DOMBuilderFilter.FILTER_SKIP
The value of the constant **DOMBuilderFilter.FILTER_SKIP** is **3**.
DOMBuilderFilter.FILTER_INTERRUPT
The value of the constant **DOMBuilderFilter.FILTER_INTERRUPT** is **4**.

Objects that implement the **DOMBuilderFilter** interface:

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

whatToShow

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

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

startElement(elt)

This function returns a **Number**.

The **elt** parameter is an object that implements the **Element** interface.

acceptNode(enode)

This function returns a **Number**.

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

Properties of the **DOMBuilder** Constructor function:

DOMBuilder.ACTION_REPLACE

The value of the constant **DOMBuilder.ACTION_REPLACE** is **1**.

DOMBuilder.ACTION_APPEND_AS_CHILDREN

The value of the constant **DOMBuilder.ACTION_APPEND_AS_CHILDREN** is **2**.

DOMBuilder.ACTION_INSERT_AFTER

The value of the constant **DOMBuilder.ACTION_INSERT_AFTER** is **3**.

DOMBuilder.ACTION_INSERT_BEFORE

The value of the constant **DOMBuilder.ACTION_INSERT_BEFORE** is **4**.

Objects that implement the **DOMBuilder** interface:

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

entityResolver

This property is an object that implements the **DOMEntityResolver** interface.

errorHandler

This property is an object that implements the **DOMErrorHandler** interface.

filter

This property is an object that implements the **DOMBuilderFilter** interface.

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

setFeature(name, state)

This function has no return value.

The **name** parameter is a **String**.

The **state** parameter is a **Boolean**.

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

canSetFeature(name, state)

This function returns a **Boolean**.

The **name** parameter is a **String**.

The **state** parameter is a **Boolean**.

getFeature(name)

This function returns a **Boolean**.

The **name** parameter is a **String**.

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

parseURI(uri)

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

The **uri** parameter is a **String**.

parse(is)

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

The **is** parameter is an object that implements the **DOMInputSource** interface.

parseWithContext(is, cnode, action)

This function has no return value.

The **is** parameter is an object that implements the **DOMInputSource** interface.

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

The **action** parameter is a **Number**.

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

Objects that implement the **DOMWriter** interface:

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

encoding

This property is a **String**.

newLine

This property is a **String**.

filter

This property is an object that implements the **DOMWriterFilter** interface.

errorHandler

This property is an object that implements the **DOMErrorHandler** interface.

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

setFeature(name, state)

This function has no return value.

The **name** parameter is a **String**.

The **state** parameter is a **Boolean**.

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

canSetFeature(name, state)

This function returns a **Boolean**.

The **name** parameter is a **String**.

The **state** parameter is a **Boolean**.

getFeature(name)

This function returns a **Boolean**.

The **name** parameter is a **String**.

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

writeNode(destination, wnode)

This function returns a **Boolean**.

The **destination** parameter is an object that implements the **Object** interface.

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

writeToString(wnode)

This function returns a **String**.

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

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

Objects that implement the **DOMWriterFilter** interface:

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

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

whatToShow

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

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.

16-bit unit
The base unit of a `DOMString`. This indicates that indexing on a `DOMString` occurs in units of 16 bits. This must not be misunderstood to mean that a `DOMString` can store arbitrary 16-bit units. A `DOMString` is a character string encoded in UTF-16; this means that the restrictions of UTF-16 as well as the other relevant restrictions on character strings must be maintained. A single character, for example in the form of a numeric character reference, may correspond to one or two 16-bit units.

API
An *API* is an Application Programming Interface, a set of functions or methods used to access some functionality.

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.67] 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.

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 document
A document is *well-formed* if it is tag valid and entities are limited to single elements (i.e., single sub-trees).

References

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

F.1: Normative references

[CharModel]
Character Model for the World Wide Web 1.0, M. D¨rst, et al., Editors. World Wide Web Consortium, January 2001. This version of the Character Model for the World Wide Web Specification is http://www.w3.org/TR/2002/WD-charmod-20020220. The latest version of Character Model is available at http://www.w3.org/TR/charmod.

[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 2002. This version of the Document Object Model Level 3 Core Specification is http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20020114. 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, December 1999. This version of the ECMAScript Language is available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM.

[ISO/IEC 10646]
ISO/IEC 10646-1993 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

[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.

[IETF RFC 2396]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt.

[IETF RFC 3023]
XML Media Types, M. Murata, S. St.Laurent, D. Kohn, Editors. Internet Engineering Task Force, January 2001. Available at http://www.ietf.org/rfc/rfc3023.txt.

[SAX]
Simple API for XML, D. Megginson and D. Brownell, Maintainers. Available at http://www.saxproject.org/

[Unicode 2.0]
The Unicode Standard, Version 2.0.. The Unicode Consortium, 1996. Reading, Mass.: Addison-Wesley Developers Press. ISBN 0-201-48345-9.

[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, J. Cowan, Editor. World Wide Web Consortium, 13 December 2001. This version of the XML 1.1 Specification is http://www.w3.org/TR/2001/WD-xml11-20011213. The latest version of XML 1.1 is available at http://www.w3.org/TR/xml11.

[XML Information set]
XML Information Set, J. Cowan and R. Tobin, Editors. World Wide Web Consortium, 24 October 2001. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2001/REC-xml-infoset-20011024. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset.

[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.

F.2: Informative references

[Canonical XML]
Canonical XML Version 1.0, J. Boyer, Editor. World Wide Web Consortium, 15 March 2001. This version of the Canonical XML Recommendation is http://www.w3.org/TR/2001/REC-xml-c14n-20010315. The latest version of Canonical XML is available at http://www.w3.org/TR/xml-c14n.

[DOM Level 3 Events]
Document Object Model Level 3 Events Specification, T. Pixley, Editor. World Wide Web Consortium, February 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.

[JAXP]
Java API for XML Processing (JAXP). Sun Microsystems. Available at http://java.sun.com/xml/xml_jaxp.html

[IETF RFC 2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[XML Schema Part 1]

XML Schema Part 1: Structures, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Part 1 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-1-20010502. The latest version of XML Schema Part 1 is available at http://www.w3.org/TR/xmlschema-1.

F.2: Informative references

Index

16-bit unit 16, 67

[attributes]

abort
acceptNode
ACTION_APPEND_AS_CHILDREN
ACTION_INSERT_AFTER
ACTION_INSERT_BEFORE
ACTION_REPLACE
API 9, 67
async

baseURI
byteStream

Canonical XML 28, 70
canSetFeature 26, 33
characterStream
CharModel 28, 69
createDOMBuilder
createDOMInputSource
createDOMWriter

document element
document order
DocumentLS
DOM Level 2 Core 28, 43, 69
DOM Level 3 Core 9, 10, 13, 15, 22, 28, 28, 69
DOM Level 3 Events 17, 70
DOMBuilder
DOMBuilderFilter
DOMEntityResolver
DOMImplementationLS
DOMInputSource
DOMInputStream
DOMOutputStream
DOMReader
DOMWriter
DOMWriterFilter

ECMAScript
encoding 16, 32
entityResolver
errorHandler 25, 32

filter 26, 32
FILTER_ACCEPT
FILTER_INTERRUPT
FILTER_REJECT
FILTER_SKIP

getFeature 26, 33

IETF RFC 2396 17, 16, 18, 27, 69 | IETF RFC 2616 16, 22, 70 | IETF RFC 3023 22, 69

inputSource 17, 17 | ISO/IEC 10646 16, 69

Java | JAXP 9, 70

load | loadXML | LSLoadEvent

LSProgressEvent

MODE_ASYNCHRONOUS | MODE_SYNCHRONOUS

newDocument | newLine

OMG IDL

parse | parseURI | parseWithContext

position | publicId

resolveEntity

saveXML | SAX 9, 18, 70 | setFeature 28, 34

startElement | stringData | systemId

tokenized | totalSize

Unicode 2.0 16, 70

well-formed document 21, 20, 28, 67
whatToShow 20, 35
writeNode
writeToString

XML 1.0 11, 16, 22, 28, 33, 67, 70
XML 1.1 22, 33, 70
XML Information set 22, 70
XML Namespaces 22, 70
XML Schema Part 1 11, 71