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

Version 1.0

W3C Working Draft 19 June 2003

This version:
http://www.w3.org/TR/2003/WD-DOM-Level-3-LS-20030619
Latest version:
http://www.w3.org/TR/DOM-Level-3-LS
Previous version:
http://www.w3.org/TR/2003/WD-DOM-Level-3-LS-20030226

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

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

Abstract

This specification defines the Document Object Model Load and Save Level 3, a platform- and language-neutral interface that allows programs and scripts to dynamically load the content of an XML document into a DOM document and serialize a DOM document into an XML document; DOM documents being defined in [DOM Level 2 Core] or newer, and XML documents being defined in [XML 1.0] or newer.

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 document contains the Document Object Model Level 3 Load and Save specification and is a Last Call Working Draft for review by W3C members and other interested parties. The review period for this document ends on *31 July 2003*. Comments 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/.

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 W3C.

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

W3C Copyright Notices and Licenses

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

W3C® Document Copyright Notice and License

Note: This section is a copy of the W3C® Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231.

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

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

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

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. *If it exists*, the STATUS of the W3C document.

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

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

requirements.

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

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

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

W3C® Software Copyright Notice and License

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

Copyright © 2003 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

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

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

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

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

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

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

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

W3C® Short Software Notice

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

Copyright © 2003 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

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

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

1. Document Object Model Load and Save

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

This section defines a set of interfaces for loading and saving document objects as defined in [DOM Level 2 Core] or newer. 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.55] also allows filtering of XML content using only DOM API calls; access and manipulation of the `Document` is defined in [DOM Level 2 Core] or newer.

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

1.1 Overview of the Interfaces

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

- `DOMImplementationLS` [p.11] -- A new `DOMImplementation` interface that provides the factory methods for creating the objects required for loading and saving.
- `DOMParser` [p.13] -- A parser interface.
- `DOMInput` [p.20] -- Encapsulate information about the XML document to be loaded.
- `DOMResourceResolver` [p.22] -- During loading, provides a way for applications to redirect references to external entities.
- `DOMParserFilter` [p.23] -- Provide the ability to examine and optionally remove Element nodes as they are being processed during the parsing of a document.
- `DOMSerializer` [p.27] -- An interface for writing out or serializing DOM documents.
- `DOMOutput` [p.33] -- Encapsulate information about the XML document to be serialized.
- `DOMSerializerFilter` [p.34] -- Provide the ability to examine and optionally remove nodes as they are being processed during the serialization of a document.
- `DocumentLS` [p.35] -- Provides a client or browser style interface for loading and saving.
- `ElementLS` [p.38] -- Provides a user convenient mechanism by which the children of an element can be serialized to a string, or replaced by the result of parsing a provided string.

1.2 Basic types

To ensure interoperability, this specification specifies the following basic types used in various DOM modules. Even though the DOM uses the basic types in the interfaces, bindings may use different types and normative bindings are only given for Java and ECMAScript in this specification.

1.2.1 The `DOMInputStream` type

This type is used to represent a sequence of input bytes.

Type Definition *DOMInputStream*

A `DOMInputStream` [p.10] represents a reference to a byte stream source of an XML input.
IDL Definition

```
typedef Object DOMInputStream;
```

Note: For Java, `DOMInputStream` [p.10] is bound to the `java.io.InputStream` type. For ECMAScript, `DOMInputStream` is bound to `Object`.

1.2.2 The `DOMOutputStream` type

This type is used to represent a sequence of output bytes.

Type Definition *DOMOutputStream*

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

```
typedef Object DOMOutputStream;
```

Note: For Java, `DOMOutputStream` [p.10] is bound to the `java.io.OutputStream` type. For ECMAScript, `DOMOutputStream` is bound to `Object`.

1.2.3 The `DOMReader` type

This type is used to represent a sequence of input characters in 16-bit units [p.55] . The encoding used for the characters is UTF-16, as defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).

Type Definition *DOMReader*

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

```
typedef Object DOMReader;
```

Note: For Java, `DOMReader` [p.10] is bound to the `java.io.Reader` type. For ECMAScript, `DOMReader` is *NOT* bound, and therefore as no recommended meaning in ECMAScript.

1.2.4 The `DOMWriter` type

This type is used to represent a sequence of output characters in 16-bit units [p.55] . The encoding used for the characters is UTF-16, as defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).

Type Definition *DOMWriter*

A `DOMWriter` [p.11] represents a character stream for the XML input.
IDL Definition

```
typedef Object DOMWriter;
```

Note: For Java, `DOMWriter` [p.11] is bound to the `java.io.Writer` type. For ECMAScript, `DOMWriter` is *NOT* bound, and therefore as no recommended meaning in ECMAScript.

1.3 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.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values `"LS"` (or `"LS-Async"`) 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 [DOM Level 2 Core].

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

For additional information about *conformance*, please see the DOM Level 3 Core specification [DOM Level 3 Core].

Interface *DOMImplementationLS*

`DOMImplementationLS` contains the factory methods for creating Load and Save objects.

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 `DOMImplementation.getFeature` with parameter values `"LS"` (or `"LS-Async"`) and `"3.0"` (respectively).
IDL Definition

```
interface DOMImplementationLS {

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

  DOMParser          createDOMParser(in unsigned short mode,
                                     in DOMString schemaType)
```

```
                                        raises(DOMException);
  DOMSerializer      createDOMSerializer();
  DOMInput           createDOMInput();
};
```

Definition group *DOMImplementationLSMode*

Integer parser mode constants.

Defined Constants

`MODE_ASYNCHRONOUS`

Create an asynchronous `DOMParser` [p.13] .

`MODE_SYNCHRONOUS`

Create a synchronous `DOMParser` [p.13] .

Methods

`createDOMInput`

Create a new empty input source.

Return Value

`DOMInput` [p.20]	The newly created input object.

No Parameters

No Exceptions

`createDOMParser`

Create a new `DOMParser` [p.13] . The newly constructed parser may then be configured by means of its `DOMConfiguration` object, 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 `DOMParser` [p.13] that is created will operate in synchronous mode, if it's `MODE_ASYNCHRONOUS` then the `DOMParser` that is created will operate in asynchronous mode.

`schemaType` of type `DOMString`

An absolute URI representing the type of the schema [p.55] language used during the load of a `Document` using the newly created `DOMParser` [p.13] . Note that no lexical checking is done on the absolute URI. In order to create a `DOMParser` for any kind of schema types (i.e. the DOMParser 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

`DOMParser` [p.13]	The newly created `DOMParser` object. This `DOMParser` is either synchronous or asynchronous depending on the value of the `mode` argument. **Note:** By default, the newly created `DOMParser` does not contain a `DOMErrorHandler`, i.e. the value of the "*error-handler*" configuration parameter is `null`. However, implementations may provide a default error handler at creation time. In that case, the initial value of the `"error-handler"` configuration parameter on the new created `DOMParser` contains a reference to the default error handler.

Exceptions

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

`createDOMSerializer`

Create a new `DOMSerializer` [p.27] object.

Return Value

`DOMSerializer` [p.27]	The newly created `DOMSerializer` object. **Note:** By default, the newly created `DOMSerializer` has no `DOMErrorHandler`, i.e. the value of the `"error-handler"` configuration parameter is `null`. However, implementations may provide a default error handler at creation time. In that case, the initial value of the `"error-handler"` configuration parameter on the new created `DOMSerializer` contains a reference to the default error handler.

No Parameters

No Exceptions

Interface *DOMParser*

An interface to an object that is able to build, or augment, a DOM tree from various input sources.

`DOMParser` provides an API for parsing XML and building the corresponding DOM document structure. A `DOMParser` instance can be obtained by invoking the `DOMImplementationLS.createDOMParser()` [p.12] method.

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

- there is only one `Text` node for each block of text. The `Text` nodes are in "normal" form: only structure (e.g. elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent 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 parameters "*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 parameters `data-type-normalization` is set to `false`, the XML 1.0 attribute normalization is guaranteed to occur, and if the attributes list does not contain namespace declarations, the `attributes` attribute on `Element` node represents the property **[attributes]** defined in [XML Information set].

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

Events supported by asynchronous `DOMParser` objects are:

load
: The `DOMParser` finishes to load the document. See also the definition of the `LSLoadEvent` [p.26] interface.

progress
: The `DOMParser` signals a progress as a document is parsed. See also the definition of the `LSProgressEvent` [p.26] interface.

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

While parsing an input source, errors are reported to the application through the error handler (`DOMParser.config` [p.16] 's "*error-handler*" parameter). This specification does in no way try to define all possible errors that can occur while parsing XML, or any other markup, but some common error cases are defined. The types (`DOMError.type`) of errors and warnings defined by this specification are:

`"unsupported-media-type" [fatal]`
: Raised if the configuration parameter "supported-media-types-only [p.17] " is set to `true` and an unsupported media type is encountered.

`"unsupported-encoding" [fatal]`
: Raised if an unsupported encoding is encountered.

`"doctype-not-allowed" [fatal]`
: Raised if the configuration parameter "disallow-doctype [p.16] " is set to `true` and a doctype is encountered.

`"unknown-character-denormalization" [fatal]`
: Raised if the configuration parameter "ignore-unknown-character-denormalizations [p.17] " is set to `false` and a character is encountered for which the processor cannot determine the normalization properties.

`"unbound-namespace-in-entity" [warning]`
: Raised if the configuration parameter "*entities*" is set to `true` and an unbound namespace prefix is encounterd in an entity declaration.

`"pi-base-uri-not-preserved" [warning]`
: Raised if a processing instruction is encoutered in a location where the base URI of the processing instruction can not be preserved.
  One example of a case where this warning will be raised is if the configuration parameter

"*entities*" is set to `false` and the following XML file is parsed:

```
<!DOCTYPE root [
<!ENTITY e SYSTEM 'subdir/myentity.ent'
]>

<root>
&e;
</root>
```

And `subdir/myentity.ent` looks like this:

```
<one>
  <two/>
</one>
<?pi 3.14159?>
<more/>
```

In addition to raising the defined errors and warnings, implementations are expected to raise implementation specific errors and warnings for any other error and warning cases such as IO errors (file not found, permission denied,...), XML well-formedness errors, and so on.

IDL Definition

```
interface DOMParser {
  readonly attribute DOMConfiguration config;
           attribute DOMParserFilter filter;
  readonly attribute boolean         async;
  readonly attribute boolean         busy;
  Document           parse(in DOMInput is)
                                     raises(DOMException);
  Document           parseURI(in DOMString uri)
                                     raises(DOMException);

  // ACTION_TYPES
  const unsigned short      ACTION_APPEND_AS_CHILDREN      = 1;
  const unsigned short      ACTION_REPLACE_CHILDREN        = 2;
  const unsigned short      ACTION_INSERT_BEFORE           = 3;
  const unsigned short      ACTION_INSERT_AFTER            = 4;
  const unsigned short      ACTION_REPLACE                 = 5;

  Node               parseWithContext(in DOMInput input,
                                      in Node context,
                                      in unsigned short action)
                                       raises(DOMException);
  void               abort();
};
```

Definition group *ACTION_TYPES*

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

Defined Constants

`ACTION_APPEND_AS_CHILDREN`

Append the result of the parse operation 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 the parse operation as the immediately following sibling of the context node. For this action to work the context node's parent must be an `Element` or a `DocumentFragment`.

`ACTION_INSERT_BEFORE`
: Insert the result of the parse operation as the immediately preceding sibling of the context node. For this action to work the context node's parent must be an `Element` or a `DocumentFragment`.

`ACTION_REPLACE`
: Replace the context node with the result of the parse operation. For this action to work, the context node must have a parent, and the parent must be an `Element` or a `DocumentFragment`.

`ACTION_REPLACE_CHILDREN`
: Replace all the children of the context node with the result of the parse operation. For this action to work, the context node must be an `Element`, a `Document`, or a `DocumentFragment`.

Attributes

`async` of type `boolean`, readonly
: `true` if the `DOMParser` is asynchronous, `false` if it is synchronous.

`busy` of type `boolean`, readonly
: `true` if the `DOMParser` is currently busy loading a document, otherwise `false`.

`config` of type `DOMConfiguration`, readonly
: The `DOMConfiguration` object used when parsing an input source. This `DOMConfiguration` is specific to the parse operation and no parameter values from this `DOMConfiguration` object are passed automatically to the `DOMConfiguration` object on the `Document` that is created, or used, by the parse operation. The DOM application is responsible for passing any needed parameter values from this `DOMConfiguration` object to the `DOMConfiguration` object referenced by the `Document` object.
  In addition to the parameters recognized in [DOM Level 3 Core], the `DOMConfiguration` objects for `DOMParser` adds or modifies the following parameters:
  `"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 section 4.3.3, "Character Encoding in Entities", in [XML 1.0]). Explicitly setting an encoding in the `DOMInput` [p.20] overrides any encoding from the protocol.
    : `false`
      : [*required*]
        The parser ignores any character set encoding information from higher-level protocols.
  `"disallow-doctype"`

`true`
[*optional*]
Throw a fatal **"doctype-not-allowed"** error if a doctype node is found while parsing the document. This is useful when dealing with things like SOAP envelopes where doctype nodes are not allowed.
`false`
[*required*] (*default*)
Allow doctype nodes in the document.
`"ignore-unknown-character-denormalizations"`
`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.
This parameter is ignored for [XML 1.0].
`false`
[*optional*]
Report an fatal **"unknown-character-denormalization"** error if a character is encountered for which the processor cannot determine the normalization properties.
`"infoset"`
See the definition of `DOMConfiguration` for a description of this parameter.
Unlike in [DOM Level 3 Core], this parameter will default to `true` for `DOMParser`.
`"namespaces"`
`true`
[*required*] (*default*)
Perform the namespace processing as defined in [XML Namespaces].
`false`
[*optional*]
Do not perform the namespace processing.
`"supported-media-types-only"`
`true`
[*optional*]
Check that the media type of the parsed resource is a supported media type. If an unsupported media type is encountered, a fatal error of type **"unsupported-media-type"** will be raised. The media types defined in [IETF RFC 3023] must always be accepted.
`false`
[*required*] (*default*)
Accept any media type.
The parameter "*well-formed*" cannot be set to `false`.
`filter` of type `DOMParserFilter` [p.23]
When a filter is provided, the implementation will call out to the filter as it is constructing the DOM tree structure. The filter can choose to remove elements from the document being constructed, or to terminate the parsing early.

The filter is invoked after the operations requested by the `DOMConfiguration` parameters have been applied. For example, if "*validate*" is set to `true`, the validation is done before invoking the filter.

Methods

`abort`

Abort the loading of the document that is currently being loaded by the `DOMParser`. If the `DOMParser` is currently not busy, a call to this method does nothing.

No Parameters

No Return Value

No Exceptions

`parse`

Parse an XML document from a resource identified by a `DOMInput` [p.20] .

Parameters

`is` of type `DOMInput` [p.20]

The `DOMInput` from which the source of the document is to be read.

Return Value

`Document`	If the `DOMParser` is a synchronous `DOMParser`, the newly created and populated `Document` is returned. If the `DOMParser` is asynchronous, `null` is returned since the document object may not yet be constructed when this method returns.

Exceptions

`DOMException`	INVALID_STATE_ERR: Raised if the `DOMParser`'s `DOMParser.busy` [p.16] attribute is `true`.

`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, future versions of this specification may define the behavior.

Parameters

`uri` of type `DOMString`

The location of the XML document to be read.

Return Value

`Document`	If the `DOMParser` is a synchronous `DOMParser`, the newly created and populated `Document` is returned. If the `DOMParser` is asynchronous, `null` is returned since the document object may not yet be constructed when this method returns.

Exceptions

`DOMException`	INVALID_STATE_ERR: Raised if the `DOMParser.busy` [p.16] attribute is `true`.

`parseWithContext`

Parse an XML fragment from a resource identified by a `DOMInput` [p.20] and insert the content into an existing document at the position specified with the `context` and `action` arguments. When parsing the input stream, the context node is used for resolving unbound namespace prefixes. The context node's `ownerDocument` node (or the node itself if the node of type `DOCUMENT_NODE`) is used to resolve default attributes and entity references.

As the new data is inserted into the document, at least one mutation event is fired per new immediate child or sibling of the context node.

If the context node is a `Document` node and the action is `ACTION_REPLACE_CHILDREN`, then the document that is passed as the context node will be changed such that it's `xmlEncoding`, `documentURI`, `xmlVersion`, `actualEncoding`, `xmlStandalone`, and all other such attributes are set to what they would be set to if the input source was parsed using `DOMParser.parse()` [p.18] .

If the `DOMParser` is asynchronous then the insertion of the resulting DOM structure is atomic, e.g. the whole structure is inserted only once the whole input stream is completely parsed without errors.

If an error occurs while parsing, the caller is notified through the `ErrorHandler` instance associated with the "*error-handler*" parameter of the `DOMConfiguration`.

When calling `parseWithContext`, the values of the following configuration parameters will be ignored and their default values will always be used instead: "*validate*", "*validate-if-schema*", and "*whitespace-in-element-content*".

Parameters

`input` of type `DOMInput` [p.20]

The `DOMInput` from which the source document is to be read. The source document must be an XML fragment, i.e. anything except a complete XML document (except in the case where the context node of type `DOCUMENT_NODE`, and the action is `ACTION_REPLACE_CHILDREN`), a DOCTYPE (internal subset), entity declaration(s), notation declaration(s), or XML or text declaration(s).

`context` 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` node, e.g. it cannot be an `Attribute` node.

`action` of type `unsigned short`

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

Return Value

`Node`	Return the node that is the result of the parse operation. If the result is more than one top-level node, the first one is returned.

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if the `DOMParser` doesn't support this method. NO_MODIFICATION_ALLOWED_ERR: Raised if the context node is a read only node [p.55] . INVALID_STATE_ERR: Raised if the `DOMParser.busy` [p.16] attribute is `true`.

Interface *DOMInput*

This interface represents an input source for data.

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), a base URI, and/or a character stream.

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

The application is expected to provide objects that implement this interface whenever such objects are needed. The application can either provide its own objects that implement this interface, or it can use the generic factory method `DOMImplementationLS.createDOMInput()` [p.12] to create objects that implement this interface.

The `DOMParser` [p.13] will use the `DOMInput` object to determine how to read data. The `DOMParser` will look at the different inputs specified in the `DOMInput` in the following order to know which one to read from, the first one through which data is available will be used:

1. `DOMInput.characterStream` [p.21]
2. `DOMInput.byteStream` [p.21]
3. `DOMInput.stringData` [p.21]
4. `DOMInput.systemId` [p.21]
5. `DOMInput.publicId` [p.21]

`DOMInput` objects belong to the application. The DOM implementation will never modify them (though it may make copies and modify the copies, if necessary).

IDL Definition

```
interface DOMInput {
  // Depending on the language binding in use,
  // this attribute may not be available.
           attribute DOMReader       characterStream;
           attribute DOMInputStream  byteStream;
           attribute DOMString       stringData;
           attribute DOMString       systemId;
           attribute DOMString       encoding;
```

```
            attribute DOMString       publicId;
            attribute DOMString       baseURI;
            attribute boolean         certified;
};
```

Attributes

`baseURI` of type `DOMString`

The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving a relative `systemId` to an absolute URI.

If, when used, the base URI is itself a relative URI, an empty string, or null, the behavior is implementation dependent.

`byteStream` of type `DOMInputStream` [p.10]

An attribute of a language and binding dependent type that represents a stream of bytes.

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 an XML declaration in the data.

`certified` of type `boolean`

If set to true, assume that the input is certified (see section 2.13 in [XML 1.1]) when parsing [XML 1.1].

`characterStream` of type `DOMReader` [p.10]

Depending on the language binding in use, this attribute may not be available.

An attribute of a language and binding dependent type that represents a stream of 16-bit units [p.55] . The application must encode the stream using UTF-16 (defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).

`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 or string data. 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. This may be mapped to an input source using an implementation dependent mechanism (such as catalogues or other mappings). The public identifier, if specified, may also be reported as part of the location information when errors are reported.

`stringData` of type `DOMString`

String data to parse. If provided, this will always be treated as a sequence of 16-bit units [p.55] (UTF-16 encoded characters).

`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, a character stream, or string data, but it is still useful to provide one, since the application will use it to resolve any relative URI’s and can include it in error messages and warnings (the `DOMParser` [p.13] will only attempt to fetch the resource identified by the URI reference only if there is no other input available in the input source).

If the application knows the character encoding of the object pointed to by the system identifier, it can set the encoding using the `encoding` attribute.

If the system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the DOM implementation will attempt to resolve the relative URI with the `baseURI` as the base, if that fails, the behavior is implementation dependent.

Interface *DOMResourceResolver*

`DOMResourceResolver` provides a way for applications to redirect references to external resources.

Applications needing to implement custom handling for external resources can implement this interface and register their implementation by setting the `resourceResolver` attribute of the `DOMParser` [p.13] .

The `DOMParser` [p.13] 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 URN's.

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

IDL Definition

```
interface DOMResourceResolver {
  DOMInput          resolveResource(in DOMString publicId,
                                    in DOMString systemId,
                                    in DOMString baseURI);
};
```

Methods

`resolveResource`

Allow the application to resolve external resources.

The `DOMParser` [p.13] will call this method before opening any external resource 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 `DOMParser` resolve the resource 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 URI's, 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 `DOMParser` [p.13] must resolve it fully before calling this method.

Parameters

`publicId` of type `DOMString`

The public identifier of the external entity being referenced, or `null` if no public identifier was supplied or if the resource is not an entity.

`systemId` of type `DOMString`

The system identifier, a URI reference [IETF RFC 2396], of the external resource being referenced.

`baseURI` of type `DOMString`
The absolute base URI of the resource being parsed, or `null` if there is no base URI.

Return Value

`DOMInput` [p.20]	A `DOMInput` 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 *DOMParserFilter*

`DOMParserFilter`s provide applications the ability to examine nodes as they are being constructed while parsing. 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 `DOMParserFilter` 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 `acceptNode` method on 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.

The `DOMParser` [p.13] ignores any exception raised in the filter.

IDL Definition

```
interface DOMParserFilter {

  // 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 element);
  unsigned short     acceptNode(in Node node);
  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 and 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 `DOMParser` [p.13] 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`, `SHOW_ENTITY`, and `SHOW_DOCUMENT_FRAGMENT` are meaningless here, those nodes will never be passed to a `DOMParserFilter`.
The constants used here are defined in [DOM Level 2 Traversal and Range].

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

`node` 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.55] .

No Exceptions

`startElement`

The parser will call this method 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

`element` 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. This return value will be ignored if `element` is the documentElement, the documentElement cannot be rejected.
- `FILTER_SKIP` if the `Element` should be rejected. All of its children are inserted in place of the rejected `Element` node. This return value will be ignored if `element` is the documentElement, the documentElement cannot be rejected nor skipped.
- `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.55] .

Returning any other values will result in unspecified behavior.

No Exceptions

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 DOMInput        input;
  readonly attribute unsigned long   position;
  readonly attribute unsigned long   totalSize;
};
```

Attributes

`input` of type `DOMInput` [p.20] , 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 *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 DOMInput        input;
};
```

Attributes

`input` of type `DOMInput` [p.20] , readonly
The input source that was parsed.

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

Interface *DOMSerializer*

`DOMSerializer` provides an API for serializing (writing) a DOM document out into XML. The XML data is written to a string or an output stream.

During serialization of XML data, namespace fixup is done 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.

`DOMSerializer` accepts any node type for serialization. For nodes of type `Document` or `Entity`, well-formed XML will be created when possible (well-formedness is guaranteed if the document or entity comes from a parse operation and is unchanged since it was created). The serialized output for these node types is either as a XML document or an External XML 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.

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

- `Document` nodes are written, including the XML declaration (unless the parameter "xml-declaration [p.30] " is set to `false`) and a DTD subset, if one exists in the DOM. Writing a `Document` node serializes the entire document.
- `Entity` nodes, when written directly by `DOMSerializer.write` [p.31] , outputs the entity expansion but no namespace fixup is done. The resulting output will be valid as an external entity.
- `EntityReference` 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 cannot be represented in the specified output encoding are handled according to the "*split-cdata-sections*" parameter.
  If the parameter is set to `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 parameter is set to `false`, unrepresentable characters in a CDATA section are reported as `"invalid-data-in-cdata-section"` 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 `Node` does not always generate a well-formed [p.55] XML document, i.e. a `DOMParser` [p.13] might throw 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 (>, ', and ") might not be used, except where needed (e.g. using > in cases such as ']]>'). Any characters that cannot be represented directly in the output character encoding are serialized as numeric character references.

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """. New line characters and other characters that cannot be represented directly in attribute values in the output character encoding are 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 `encoding="us-ascii"`.

When requested by setting the parameter "*normalize-characters*" on `DOMSerializer` to true, character normalization is performed according to the rules defined in [CharModel] on all data to be serialized, both markup and character data. The character 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.

When outputting unicode data, whether or not a byte order mark is serialized, or if the output is big-endian or little-endian, is implementation dependent.

Namespaces are fixed up during serialization, the serialization process will verify that namespace declarations, namespace prefixes and the namespace URI's associated with elements and attributes are consistent. If inconsistencies are found, the serialized form of the document will be altered to remove them. The method used for doing the namespace fixup while serializing a document is the algorithm defined in Appendix B.1, "Namespace normalization", of [DOM Level 3 Core].

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.
Issue DOMSerializer-change-DOM:

> We may take back what we say in the above paragraph depending on feedback from implementors, but for now the belief is that the DOM's view of the document is not changed during serialization.

While serializing a document, the parameter "discard-default-content [p.29] " controls whether or not non-specified data is serialized.

While serializing, errors are reported to the application through the error handler (`DOMSerializer.config` [p.29] 's "*error-handler*" parameter). This specification does in no way try to define all possible errors that can occur while serializing a DOM node, but some common

error cases are defined. The types (`DOMError.type`) of errors and warnings defined by this specification are:

`"invalid-data-in-cdata-section" [fatal]`
: Raised if the configuration parameter "*split-cdata-sections*" is set to `false` and invalid data is encountered in a CDATA section.

`"unsupported-encoding" [fatal]`
: Raised if an unsupported encoding is encountered.

`"unbound-namespace-in-entity" [warning]`
: Raised if the configuration parameter "*entities*" is set to `true` and an unbound namespace prefix is encounterd in a referenced entity.

`"no-output-specified" [fatal]`
: Raised when writing to a `DOMOutput` [p.33] if no output is specified in the `DOMOutput`.

In addition to raising the defined errors and warnings, implementations are expected to raise implementation specific errors and warnings for any other error and warning cases such as IO errors (file not found, permission denied,...) and so on.

IDL Definition

```
interface DOMSerializer {
  readonly attribute DOMConfiguration config;
           attribute DOMString       newLine;
           attribute DOMSerializerFilter filter;
  boolean            write(in Node node,
                           in DOMOutput destination);
  boolean            writeURI(in Node node,
                              in DOMString URI);
  DOMString          writeToString(in Node node)
                                   raises(DOMException);
};
```

Attributes

`config` of type `DOMConfiguration`, readonly
: The `DOMConfiguration` object used by the `DOMSerializer` when serializing a DOM node.
: In addition to the parameters recognized in the [DOM Level 3 Core], the `DOMConfiguration` objects for `DOMSerializer` adds, or modifies, the following parameters:

`"canonical-form"`

`true`
: [*optional*]
: This formatting writes the document according to the rules specified in [Canonical XML]. Setting this parameter to `true` will set the parameter "format-pretty-print [p.30] " to `false`.

`false`
: [*required*] (*default*)
: Do not canonicalize the output.

`"discard-default-content"`

`true`
[*required*] (*default*)
Use the `Attr.specified` attribute to decide what attributes should be discarded. Note that some implementations might use whatever information available to the implementation (i.e. XML schema, DTD, the `Attr.specified` attribute, and so on) to determine what attributes and content to discard if this parameter is set to `true`.
`false`
[*required*]
Keep all attributes and all content.

`"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. Pretty-printing changes the content of the document and may affect the validity of the document, validating implementations should preserve validity. Setting this parameter to `true` will set the parameter "*canonical-form*" to `false`.
`false`
[*required*] (*default*)
Don’t pretty-print the result.

`"ignore-unknown-character-denormalizations"`
`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 raise a `"unknown-character-denormalization"` warning (instead of raising an error, if this parameter is not set) and ignore any possible denormalizations caused by these characters.
Issue DOMSerializer-iucd-issue:
IMO it would make sense to move this parameter into the DOM Level 3 Core spec, and the error/warning should be defined there.
`false`
[*optional*]
Report an fatal error if a character is encountered for which the processor cannot determine the normalization properties.

`"normalize-characters"`
This parameter is equivalent to the one defined by `DOMConfiguration` in [DOM Level 3 Core]. Unlike in the Core, the default value for this parameter is `true`. While DOM implementations are not required to support fully normalizing the characters in the document according to the rules defined in [CharModel] supplemented by the definitions of relevant constructs from Section 2.13 of [XML 1.1], this parameter must be activated by default if supported.

`"xml-declaration"`

`true`
[*required*] (*default*)
If a `Document`, `Element`, or `Entity` node is serialized, the XML declaration, or text declaration, should be included. The version (`Document.xmlVersion` if the document is a Level 3 document, and the version is non-null, otherwise use the value "1.0"), and possibly an encoding (`DOMSerializer.encoding`, or `Document.actualEncoding` or `Document.xmlEncoding` if the document is a Level 3 document) is specified in the serialized XML declaration.

`false`
[*required*]
Do not serialize the XML and text declarations. Report a `"xml-declaration-needed"` warning if this will cause problems (i.e. the serialized data is of an XML version other than [XML 1.0], or an encoding would be needed to be able to re-parse the serialized data).

The parameters "*well-formed*", "*namespaces*", and "*namespace-declarations*" cannot be set to `false`.

`filter` of type `DOMSerializerFilter` [p.34]
When the application provides a filter, the serializer will call out to the filter before serializing each Node. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.
The filter is invoked before the operations requested by the `DOMConfiguration` parameters have been applied. For example, CDATA sections are passed to the filter even if "*cdata-sections*" is set to `false`.

`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 section 2.11, "End-of-Line Handling" in [XML 1.0], if the serialized content is XML 1.0 or section 2.11, "End-of-Line Handling" in [XML 1.1], 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

`write`
Serialize the specified node as described above in the general description of the `DOMSerializer` interface. The output is written to the supplied `DOMOutput` [p.33] .

When writing to a `DOMOutput` [p.33] , the encoding is found by looking at the encoding information that is reachable through the `DOMOutput` and the item to be written (or its owner document) in this order:

1. `DOMOutput.encoding` [p.34] ,
2. `Document.actualEncoding`,
3. `Document.xmlEncoding`.

If no encoding is reachable through the above properties, a default encoding of "UTF-8" will be used.
If the specified encoding is not supported an "unsupported-encoding" error is raised.
If no output is specified in the `DOMOutput` [p.33] , a "no-output-specified" error is raised.

Parameters

`node` of type `Node`
The node to serialize.

`destination` of type `DOMOutput` [p.33]
The destination for the serialized DOM.

Return Value

`boolean`	Returns `true` if `node` was successfully serialized and `false` in case the node couldn’t be serialized.

No Exceptions

`writeToString`

Serialize the specified node as described above in the general description of the `DOMSerializer` interface. The output is written to a `DOMString` that is returned to the caller (this method completely ignores all the encoding information available).

Parameters

`node` of type `Node`
The node to serialize.

Return Value

`DOMString`	Returns the serialized data, or `null` in case the node couldn’t be serialized.

Exceptions

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

`writeURI`

Serialize the specified node as described above in the general description of the `DOMSerializer` interface. The output is written to the supplied URI.
When writing to a URI, the encoding is found by looking at the encoding information that is reachable through the item to be written (or its owner document) in this order:

1. `Document.actualEncoding`,

2. `Document.xmlEncoding`.

If no encoding is reachable through the above properties, a default encoding of "UTF-8" will be used.
If the specified encoding is not supported an "unsupported-encoding" error is raised.

Parameters

`node` of type `Node`
: The node to serialize.

`URI` of type `DOMString`
: The URI to write to.

Return Value

`boolean` Returns `true` if `node` was successfully serialized and `false` in case the node couldn’t be serialized.

No Exceptions

Interface *DOMOutput*

This interface represents an output destination for data.

This interface allows an application to encapsulate information about an output destination in a single object, which may include a URI, a byte stream (possibly with a specified encoding), a base URI, and/or a character stream.

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

The application is expected to provide objects that implement this interface whenever such objects are needed. The application can either provide its own objects that implement this interface, or it can use the generic factory method `DOMImplementationLS.createDOMOutput()` to create objects that implement this interface.

The `DOMSerializer` [p.27] will use the `DOMOutput` object to determine where to serialize the output to. The `DOMSerializer` will look at the different outputs specified in the `DOMOutput` in the following order to know which one to output to, the first one that data can be output to will be used:

1. `DOMOutput.characterStream` [p.34]
2. `DOMOutput.byteStream` [p.34]
3. `DOMOutput.systemId` [p.34]

`DOMOutput` objects belong to the application. The DOM implementation will never modify them (though it may make copies and modify the copies, if necessary).

IDL Definition

```
interface DOMOutput {
  // Depending on the language binding in use,
  // this attribute may not be available.
           attribute DOMWriter       characterStream;
           attribute DOMOutputStream byteStream;
           attribute DOMString       systemId;
           attribute DOMString       encoding;
};
```

Attributes

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

An attribute of a language and binding dependent type that represents a writable stream of bytes.

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 an XML declaration in the data.

`characterStream` of type `DOMWriter` [p.11]

Depending on the language binding in use, this attribute may not be available.

An attribute of a language and binding dependent type that represents a writable stream to which 16-bit units [p.55] can be output. The application must encode the stream using UTF-16 (defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).

`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 or string data. 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].

`systemId` of type `DOMString`

The system identifier, a URI reference [IETF RFC 2396], for this output destination.

If the application knows the character encoding of the object pointed to by the system identifier, it can set the encoding using 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 *DOMSerializerFilter*

`DOMSerializerFilter`s provide applications the ability to examine nodes as they are being serialized and decide what nodes should be serialized or not. The `DOMSerializerFilter` interface is based on the `NodeFilter` interface defined in [DOM Level 2 Traversal and Range].

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

The result of any attempt to modify a node passed to a `DOMSerializerFilter` is implementation dependent.

IDL Definition

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

Attributes

`whatToShow` of type `unsigned long`, readonly

Tells the `DOMSerializer` [p.27] 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`, `SHOW_ATTRIBUTE`, and `SHOW_DOCUMENT_FRAGMENT` are meaningless here, those nodes will never be passed to a `DOMSerializerFilter`.
The constants used here are defined in [DOM Level 2 Traversal and Range].

1.4 Convenience Interfaces

The interfaces defined in this section provide no direct functionality that cannot be achieved with the load and save interfaces defined in the earlier sections of this specification. These interfaces are defined for developer convenience only, and supporting them is optional.

These interfaces are not nearly as flexible as the ones defined earlier in this specification, for example, no configuration parameters are settable when calling these methods, and the values of all configuration parameters are predefined.

Interface *DocumentLS*

The `DocumentLS` interface provides a mechanism by which the content of a document can be serialized, or replaced with the DOM tree produced when loading a URI, or parsing a string.

If the `DocumentLS` interface is supported, 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 by using the method `Node.getFeature` with parameter values `"DocumentLS"` and `"3.0"` (respectively) on an `Document`, if the `Document` supports the feature `"Core"` version `"3.0"` defined in [DOM Level 3 Core]

This interface is optional. If supported, implementations must support version `"3.0"` of the feature `"DocumentLS"`.

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 node)
                                        raises(DOMException);
};
```

Attributes

`async` of type `boolean`

Indicates whether the method `DocumentLS.load()` [p.36] 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 `true`.

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 a "load" event (as defined in [DOM Level 3 Events], except that the `Event.targetNode` will be the document, not an element) will be dispatched on the document. If an error occurs, an implementation dependent "error" event will be dispatched on the document. 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 parameters used in the `DOMParser` [p.13] interface are assumed to have their default values with the exception that the parameters `"entities"`, `"normalize-characters"`, `"check-character-normalization"` are set to `"false"`.

The result of a call to this method is the same the result of a call to `DOMParser.parseWithContext` [p.19] with an input stream referencing the URI that was passed to this call, the document as the context node, and the action `ACTION_REPLACE_CHILDREN`.

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 UTF-16. All other encoding information is ignored.
The parameters used in the `DOMParser` [p.13] interface are assumed to have their default values when invoking this method.
The result of a call to this method is the same the result of a call to `DOMParser.parseWithContext` [p.19] with an input stream containing the string passed to this call, the document as the context node, and the action `ACTION_REPLACE_CHILDREN`.

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 parameters used in the `DOMSerializer` [p.27] interface are assumed to have their default values when invoking this method.
The result of a call to this method is the same the result of a call to `DOMSerializer.writeToString` [p.32] with the document as the node to write.

Parameters

`node` 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 occurred.

Exceptions

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

Interface *ElementLS*

The `ElementLS` interface provides a convenient mechanism by which the children of an element can be serialized to a string, or replaced by the result of parsing a provided string.

If the `ElementLS` interface is supported, the expectation is that an instance of the `ElementLS` interface can be obtained by using binding-specific casting methods on an instance of the `Element` interface, or by using the method `Node.getFeature` with parameter values `"ElementLS"` and `"3.0"` (respectively) on an `Element`, if the `Element` supports the feature `"Core"` version `"3.0"` defined in [DOM Level 3 Core].

This interface is optional. If supported, implementations must support version `"3.0"` of the feature `"ElementLS"`.

IDL Definition

```
interface ElementLS {
          attribute DOMString       markupContent;
};
```

Attributes

`markupContent` of type `DOMString`

The content of the element in serialized form.

When getting the value of this attribute, the children are serialized in document order and the serialized result is returned. This is equivalent of calling `DOMSerializer.writeToString()` [p.32] on all children in document order and appending the result of the individual results to a single string that is then returned as the value of this attribute.

When setting the value of this attribute, all children of the element are removed, the provided string is parsed and the result of the parse operation is inserted into the element. This is equivalent of calling `DOMParser.parseWithContext()` [p.19] passing in the provided string (through the input source argument), the `Element`, and the action `ACTION_REPLACE_CHILDREN`. If an error occurs while parsing the provided string, the `Element`'s owner document's error handler will be called, and the `Element` is left with no children.

Both setting and getting the value of this attribute assumes that the parameters in the `DOMConfiguration` object have their default values.

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/2003/WD-DOM-Level-3-LS-20030619/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   Object DOMWriter;

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

  interface DOMParser;
  interface DOMSerializer;
  interface DOMInput;
  interface DOMParserFilter;
  interface DOMSerializerFilter;
  interface DOMOutput;

  interface DOMImplementationLS {

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

    DOMParser          createDOMParser(in unsigned short mode,
                                       in DOMString schemaType)
                                        raises(dom::DOMException);
    DOMSerializer      createDOMSerializer();
    DOMInput           createDOMInput();
```

```
};

interface DOMParser {
  readonly attribute DOMConfiguration config;
           attribute DOMParserFilter filter;
  readonly attribute boolean         async;
  readonly attribute boolean         busy;
  Document           parse(in DOMInput is)
                                  raises(dom::DOMException);
  Document           parseURI(in DOMString uri)
                                  raises(dom::DOMException);

  // ACTION_TYPES
  const unsigned short      ACTION_APPEND_AS_CHILDREN      = 1;
  const unsigned short      ACTION_REPLACE_CHILDREN        = 2;
  const unsigned short      ACTION_INSERT_BEFORE           = 3;
  const unsigned short      ACTION_INSERT_AFTER            = 4;
  const unsigned short      ACTION_REPLACE                 = 5;

  Node               parseWithContext(in DOMInput input,
                                      in Node context,
                                      in unsigned short action)
                                      raises(dom::DOMException);
  void               abort();
};

interface DOMInput {
  // Depending on the language binding in use,
  // this attribute may not be available.
           attribute DOMReader       characterStream;
           attribute DOMInputStream  byteStream;
           attribute DOMString       stringData;
           attribute DOMString       systemId;
           attribute DOMString       encoding;
           attribute DOMString       publicId;
           attribute DOMString       baseURI;
           attribute boolean         certified;
};

interface DOMResourceResolver {
  DOMInput           resolveResource(in DOMString publicId,
                                     in DOMString systemId,
                                     in DOMString baseURI);
};

interface DOMParserFilter {

  // 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 element);
  unsigned short     acceptNode(in Node node);
  readonly attribute unsigned long   whatToShow;
};
```

```
  interface DOMSerializer {
    readonly attribute DOMConfiguration config;
             attribute DOMString       newLine;
             attribute DOMSerializerFilter filter;
    boolean            write(in Node node,
                             in DOMOutput destination);
    boolean            writeURI(in Node node,
                                in DOMString URI);
    DOMString          writeToString(in Node node)
                                     raises(dom::DOMException);
  };

  interface DOMOutput {
    // Depending on the language binding in use,
    // this attribute may not be available.
             attribute DOMWriter       characterStream;
             attribute DOMOutputStream byteStream;
             attribute DOMString       systemId;
             attribute DOMString       encoding;
  };

  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 node)
                                     raises(dom::DOMException);
  };

  interface ElementLS {
             attribute DOMString       markupContent;
  };

  interface LSProgressEvent : events::Event {
    readonly attribute DOMInput        input;
    readonly attribute unsigned long   position;
    readonly attribute unsigned long   totalSize;
  };

  interface LSLoadEvent : events::Event {
    readonly attribute Document        newDocument;
    readonly attribute DOMInput        input;
  };

  interface DOMSerializerFilter : 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/2003/WD-DOM-Level-3-LS-20030619/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 DOMParser createDOMParser(short mode,
                                     String schemaType)
                                     throws DOMException;

    public DOMSerializer createDOMSerializer();

    public DOMInput createDOMInput();

}
```

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

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

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

public interface DOMParser {
    public DOMConfiguration getConfig();

    public DOMParserFilter getFilter();
    public void setFilter(DOMParserFilter filter);

    public boolean getAsync();

    public boolean getBusy();

    public Document parse(DOMInput is)
                          throws DOMException;

    public Document parseURI(String uri)
                             throws DOMException;
```

```
    // ACTION_TYPES
    public static final short ACTION_APPEND_AS_CHILDREN = 1;
    public static final short ACTION_REPLACE_CHILDREN   = 2;
    public static final short ACTION_INSERT_BEFORE      = 3;
    public static final short ACTION_INSERT_AFTER       = 4;
    public static final short ACTION_REPLACE            = 5;

    public Node parseWithContext(DOMInput input,
                                 Node context,
                                 short action)
                                 throws DOMException;

    public void abort();

}
```

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

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

import org.w3c.dom.DOMInputStream;
import org.w3c.dom.DOMReader;

public interface DOMInput {
    public DOMReader getCharacterStream();
    public void setCharacterStream(DOMReader characterStream);

    public DOMInputStream getByteStream();
    public void setByteStream(DOMInputStream byteStream);

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

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

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

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

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

    public boolean getCertified();
    public void setCertified(boolean certified);

}
```

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

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

public interface DOMResourceResolver {
    public DOMInput resolveResource(String publicId,
                                    String systemId,
                                    String baseURI);

}
```

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

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

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

public interface DOMParserFilter {
    // 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 element);

    public short acceptNode(Node node);

    public int getWhatToShow();

}
```

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

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

import org.w3c.dom.events.Event;

public interface LSProgressEvent extends Event {
    public DOMInput getInput();

    public int getPosition();

    public int getTotalSize();

}
```

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 DOMInput getInput();

}
```

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

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

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

public interface DOMSerializer {
    public DOMConfiguration getConfig();

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

    public DOMSerializerFilter getFilter();
    public void setFilter(DOMSerializerFilter filter);

    public boolean write(Node node,
                         DOMOutput destination);

    public boolean writeURI(Node node,
                            String URI);

    public String writeToString(Node node)
                                throws DOMException;

}
```

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

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

import org.w3c.dom.DOMWriter;
import org.w3c.dom.DOMOutputStream;

public interface DOMOutput {
    public DOMWriter getCharacterStream();
    public void setCharacterStream(DOMWriter characterStream);

    public DOMOutputStream getByteStream();
    public void setByteStream(DOMOutputStream byteStream);

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

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

}
```

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

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

import org.w3c.dom.traversal.NodeFilter;

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

}
```

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 node)
                          throws DOMException;

}
```

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

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

public interface ElementLS {
    public String getMarkupContent();
    public void setMarkupContent(String markupContent);

}
```

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:

createDOMParser(mode, schemaType)

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

The **mode** parameter is a **Number**.

The **schemaType** parameter is a **String**.

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

createDOMSerializer()

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

createDOMInput()

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

Properties of the **DOMParser** Constructor function:

DOMParser.ACTION_APPEND_AS_CHILDREN

The value of the constant **DOMParser.ACTION_APPEND_AS_CHILDREN** is **1**.

DOMParser.ACTION_REPLACE_CHILDREN

The value of the constant **DOMParser.ACTION_REPLACE_CHILDREN** is **2**.

DOMParser.ACTION_INSERT_BEFORE

The value of the constant **DOMParser.ACTION_INSERT_BEFORE** is **3**.

DOMParser.ACTION_INSERT_AFTER

The value of the constant **DOMParser.ACTION_INSERT_AFTER** is **4**.

DOMParser.ACTION_REPLACE

The value of the constant **DOMParser.ACTION_REPLACE** is **5**.

Objects that implement the **DOMParser** interface:

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

config

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

filter

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

async

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

busy

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

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

parse(is)

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

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

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**.

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

parseWithContext(input, context, action)

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

The **input** parameter is an object that implements the **DOMInput** interface.

The **context** 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.

abort()

This function has no return value.

Objects that implement the **DOMInput** interface:

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

byteStream

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

stringData

This property is a **String**.

systemId

This property is a **String**.

encoding

This property is a **String**.

publicId

This property is a **String**.

baseURI

This property is a **String**.

certified

This property is a **Boolean**.

Objects that implement the **DOMResourceResolver** interface:

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

resolveResource(publicId, systemId, baseURI)

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

The **publicId** parameter is a **String**.

The **systemId** parameter is a **String**.

The **baseURI** parameter is a **String**.

Properties of the **DOMParserFilter** Constructor function:

DOMParserFilter.FILTER_ACCEPT

The value of the constant **DOMParserFilter.FILTER_ACCEPT** is **1**.

DOMParserFilter.FILTER_REJECT

The value of the constant **DOMParserFilter.FILTER_REJECT** is **2**.

DOMParserFilter.FILTER_SKIP

The value of the constant **DOMParserFilter.FILTER_SKIP** is **3**.

DOMParserFilter.FILTER_INTERRUPT

The value of the constant **DOMParserFilter.FILTER_INTERRUPT** is **4**.

Objects that implement the **DOMParserFilter** interface:

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

whatToShow

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

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

startElement(element)

This function returns a **Number**.

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

acceptNode(node)

This function returns a **Number**.

The **node** parameter is an object that implements the **Node** 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:

input

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

position

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

totalSize

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

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.

input

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

Objects that implement the **DOMSerializer** interface:

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

config

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

newLine

This property is a **String**.

filter

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

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

write(node, destination)

This function returns a **Boolean**.

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

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

writeURI(node, URI)

This function returns a **Boolean**.

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

The **URI** parameter is a **String**.

writeToString(node)

This function returns a **String**.

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

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

Objects that implement the **DOMOutput** interface:

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

byteStream

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

systemId

This property is a **String**.

encoding

This property is a **String**.

Objects that implement the **DOMSerializerFilter** interface:

Objects that implement the **DOMSerializerFilter** 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 **DOMSerializerFilter** interface:

whatToShow

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

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(node)

This function returns a **String**.

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

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

Objects that implement the **ElementLS** interface:

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

markupContent

This property is a **String**.

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 and invited expert), Don Park (invited), Elena Litani (IBM), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Jeroen van Rotterdam (X-Hive Corporation), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape/AOL), Jon Ferraiolo (Adobe), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Lauren Wood (SoftQuad Software Inc., *former Chair*), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mary Brady (NIST), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégaret (W3C, *W3C team contact and former Chair*), Ramesh Lekshmynarayanan (Merrill Lynch), Ray Whitmer (iMall, Excite@Home, and Netscape/AOL, *Chair*), Rezaur Rahman (Intel), Rich Rollman (Microsoft), Rick Gessner (Netscape), Rick Jelliffe (invited), Rob Relyea (Microsoft), Scott Isaacs (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tim Yu (Oracle), Tom Pixley (Netscape/AOL), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

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

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

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.

namespace well-formed
: A node is a *namespace well-formed* XML node if it is a well-formed [p.55] node, and follow the productions and namespace constraints. If [XML 1.0] is used, the constraints are defined in [XML Namespaces]. If [XML 1.1] is used, the constraints are defined in [XML Namespaces 1.1].

read only node
: A *read only node* is a node that is immutable. This means its list of children, its content, and its attributes, when it is an element, cannot be changed in any way. However, a read only node can possibly be moved, when it is not itself contained in a read only node.

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

well-formed
: A node is a *well-formed* XML node if its serialized form, without doing any transformation during its serialization, matches its respective production in [XML 1.0] or [XML 1.1] (depending on the XML version in use) with all well-formedness constraints related to that production, and if the entities which are referenced within the node are also well-formed. If namespaces for XML are in use, the node must also be namespace well-formed [p.55] .

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. Dürst, F. Yergeau, R. Ishida, M. Wolf, A. Freytag, T. Texin, Editors. World Wide Web Consortium, April 2002. This version of the Character Model for the World Wide Web Specification is http://www.w3.org/TR/2002/WD-charmod-20020430. 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, June 2003. This version of the Document Object Model Level 3 Core Specification is http://www.w3.org/TR/2003/WD-DOM-Level-3-Core-20030609. The latest version of DOM Level 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core.

[DOM Level 3 Events]
Document Object Model Level 3 Events Specification, P. Le Hégaret, T. Pixley, Editors. World Wide Web Consortium, March 2003. This version of the Document Object Model Level 3 Events Specification is http://www.w3.org/TR/2003/WD-DOM-Level-3-Events-20030331. The latest version of Document Object Model Level 3 Events is available at http://www.w3.org/TR/DOM-Level-3-Events.

[DOM Level 2 Traversal and Range]
Document Object Model Level 2 Traversal and Range Specification, J. Kesselman, J. Robie, M. Champion, P. Sharpe, V. Apparao, L. Wood, Editors. World Wide Web Consortium, 13 November 2000. This version of the Document Object Model Level 2 Traversal and Range Recommendation is http://www.w3.org/TR/2000/REC-DOM-Level-2-Traversal-Range-20001113. The latest version of Document Object Model Level 2 Traversal and Range is available at http://www.w3.org/TR/DOM-Level-2-Traversal-Range.

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

[ISO/IEC 10646]
ISO/IEC 10646-2000 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. [Geneva]: International Organization for Standardization, 2000. See also International Organization for Standardization, available at http://www.iso.ch, for the latest version.

[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, and 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]

The Unicode Standard, Version 3, ISBN 0-201-61633-5, as updated from time to time by the publication of new versions. The Unicode Consortium, 2000. See also Versions of the Unicode Standard, available at http://www.unicode.org/unicode/standard/versions, for latest version and additional information on versions of the standard and of the Unicode Character Database.

[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, October 2002. This version of the XML 1.1 Specification is http://www.w3.org/TR/2002/CR-xml11-20021015. 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.

[XML Namespaces 1.1]

Namespaces in XML 1.1, T. Bray, D. Hollander, A. Layman, and R. Tobin, Editors. World Wide Web Consortium, December 2002. This version of the XML Information Set Specification is http://www.w3.org/TR/2002/CR-xml-names11-20021218/. The latest version of Namespaces in

XML is available at http://www.w3.org/TR/xml-names11/.

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.

[JAXP]

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

[IETF RFC 2616]

Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, et al., 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