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

Version 1.0

W3C Working Draft 25 October 2001

This version:
http://www.w3.org/TR/2001/WD-DOM-Level-3-ASLS-20011025
(PostScript file , PDF file , plain text , ZIP file , single HTML file)
Latest version:
http://www.w3.org/TR/DOM-Level-3-ASLS
Previous version:
http://www.w3.org/TR/2001/WD-DOM-Level-3-ASLS-20010607

Editors:
Ben Chang, *Oracle*
Jeroen van Rotterdam, *X-Hive Corporation*
Johnny Stenback, *Netscape*
Andy Heninger, *IBM (until March 2001)*
Joe Kesselman, *IBM (until September 2001)*
Rezaur Rahman, *Intel Corporation (until July 2001)*

Abstract

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

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.

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.

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

Copyright Notice

W3C Document Copyright Notice and License

W3C Software Copyright Notice and License

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

1. Abstract Schemas Object Model

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

1.1. Overview

This chapter describes the optional DOM Level 3 Abstract Schemas (AS) module. This module provides a representation for *XML abstract schemas*, e.g., DTDs [XML] and XML Schemas [XML Schema Part 0], together with operations on the abstract schemas, and how such information within the abstract schemas could be applied to *XML* [p.136] documents used in both the document-editing and AS-editing worlds. A DOM application can use the `hasFeature` method of the `DOMImplementation` interface defined in DOM Core to determine whether a given DOM supports these capabilities or not. One feature string for the AS-editing interfaces listed in this section is "`AS-EDIT`" and another feature string for document-editing interfaces is "`AS-DOC`".

This chapter interacts strongly with Document Object Model Load and Save [p.53] . Not only will that code serialize/deserialize abstract schemas, but it may also wind up defining its well-formedness and validity checks in terms of what is defined in this chapter. In addition, the AS and Load/Save functional areas uses the error-reporting mechanism allowing user-registered error callbacks introduced in [DOM Level 3 Core]. Note that this may not imply that the parser actually calls the DOM's validation code -- it may be able to achieve better performance via its own -- but the appearance to the user should probably be "as if" the DOM has been asked to validate the document, and parsers should probably be able to validate newly loaded documents in terms of a previously loaded DOM AS.

Finally, this chapter will have separate sections to address the needs of the document-editing and AS-editing worlds, along with a section that details overlapping areas such as validation. In this manner, the document-editing world's focuses on editing aspects and usage of information in the AS are made distinct from the AS-editing world's focuses on defining and manipulating the information in the AS.

1.1.1. General Characteristics

In the October 9, 1997 DOM requirements document, the following appeared: "There will be a way to determine the presence of a DTD. There will be a way to add, remove, and change declarations in the underlying DTD (if available). There will be a way to test conformance of all or part of the given document against a DTD (if available)." In later discussions, the following was added, "There will be a way to query element/attribute (and maybe other) declarations in the underlying DTD (if available)," supplementing the primitive support for these in Level 1.

That work was deferred past Level 2, in the hope that XML Schemas would be addressed as well. It is anticipated that lowest common denominator general APIs generated in this chapter can support both DTDs and XML Schemas, and other XML abstract schemas down the road.

The kinds of information that an Abstract Schema must make available are mostly self-evident from the definitions of Infoset, DTDs, and XML Schemas. Note that some kinds of information on which the DOM already relies, e.g., default values for attributes, will finally be given a visible representation here.

1.1.2. Use Cases and Requirements

The abstract schema referenced in these use cases/requirements is an abstraction and does not refer solely to DTDs or XML Schemas.

For the AS-editing and document-editing worlds, the following use cases and requirements are common to both and could be labeled as the "Validation and Other Common Functionality" section:

Use Cases:

1. CU1. Associating an abstract schema with a document, or changing the current association.
2. CU2. Using the same abstract schema with several documents, without having to reload it.

Requirements:

1. CR1. Validate against the abstract schema.
2. CR2. Retrieve information from abstract schema.
3. CR3. Load an existing abstract schema, perhaps independently from a document.
4. CR4. Being able to determine if a document has an abstract schema associated with it.
5. CR5. Associate an AS with a document and make it the active AS.

Specific to the AS-editing world, the following are use cases and requirements and could be labeled as the "AS-editing" section:

Use Cases:

1. ASU1. Clone/map all or parts of an existing abstract schema to a new or existing abstract schema.
2. ASU2. Save an abstract schema in a separate file. For example, if a DTD can be broken up into reusable pieces, which are then brought in via entity references, these can then be saved in a separate file. Note that a DTD, which may include both an internal and external subset, would be an example of an abstract schema.
3. ASU3. Modify an existing abstract schema.
4. ASU4. Create a new abstract schema.

Requirements:

1. ASR1. View and modify all parts of the abstract schema.
2. ASR2. Validate the abstract schema itself.
3. ASR3. Serialize the abstract schema.
4. ASR4. Clone all or parts of an existing abstract schema.
5. ASR5. Create a new abstract schema object.
6. ASR6. Validate portions of the XML document against the abstract schema.

Specific to the document-editing world, the following are use cases and requirements and could be labeled as the "Document-editing" section:

Use Cases:

1. DU1. For editing documents with an associated abstract schema, provide the guidance necessary so that valid documents can be modified and remain valid.
2. DU2. For editing documents with an associated abstract schema, provide the guidance necessary to transform an invalid document into a valid one.

Requirements:

1. DR1. Be able to determine if the document is well-formed, and if not, be given enough guidance to locate the error.
2. DR2. Be able to determine if the document is namespace well-formed, and if not, be given enough guidance to locate the error.
3. DR3. Be able to determine if the document is valid with respect to its associated abstract schema.
4. DR4. Be able to determine if specific modifications to a document would make it become invalid.
5. DR5. Retrieve information from all abstract schemas. One example might be getting a list of all the defined element names for document editing purposes.

General Issues:

1. I1. Some concerns exist regarding whether a single abstract Abstract Schema structure can successfully represent both namespace-unaware, e.g., DTD, and namespace-aware, e.g., XML Schema, models of document's content. For example, when you ask what elements can be inserted in a specific place, the former will report the element's `QName`, e.g., `foo:bar`, whereas the latter will report its namespace and local name, e.g., `{http://my.namespace}bar`. We have added the `isNamespaceAware` attribute to the generic AS object to help applications determine which of these fields are important, but we are still analyzing this challenge.
2. I2. RESOLVED: An XML document may be associated with multiple ASs. We have decided that only one of these is "active" (for validation and guidance) at a time. DOM applications may switch which AS is active, remove ASs that are no longer relevant, or add ASs to the list. If it becomes necessary to simultaneously consult more than one AS, it should be possible to write a "union" AS which provides that capability within this framework.
3. I3. WON'T DEAL W/THIS: Round-trippability for include/ignore statements and other constructs such as parameter entities, e.g., "macro-like" constructs, will not be supported since no data representation exists to support these constructs without having to re-parse them.
4. I4. RESOLVED: Basic interface for a common error handler for both AS and Load/Save. Agreement has been to utilize user-registered callbacks but other details to be worked out. Moved to a separate chapter by Load/Save team.
5. I5. RESOLVED: Add the ability to cache/edit an imported abstract schema instead of loading it every time, i.e., don't want to include the abstract schema every time. Implementations can do this without having this formalized though.
6. I6. Add a read-only feature string AS-QUERY, along with query methods on the abstract schema. In more detail, there are methods that let you *query* the schema as well as those that let you modify the schema and these should be a feature, i.e., AS-QUERY: Abstract Schema objects with query

interfaces.

7. I7. RESOLVED: Have the `NodeEditAS.can*()`, `CharacterDataEditAS.can*()` , and `ElementEditAS.can*()` methods throw exceptions like the `isNodeValid()` method. Resolution: no exceptions should be thrown; it should be allowed if it's not forbidden. Better descriptions are in order for the true/false returns.
8. I8. RESOLVED: Rename the document-editing interfaces so they should have uniform names such as NodeEditAS, DocumentEditAS, ElementEditAS, etc.
9. I9. RESOLVED: Remove the ASDOMStringList interface; create a new interface for document editing, which is a slimmed down version of ElementEditAS; add a slimmed down method to get an ElementEditAS. Elena to examine.
10. I10. RESOLVED: If another `ASModel` [p.15] is activated, will there be cleanup done to remove the previous ASModel's default attributes and entity definitions, if any? AS ET members felt that whatever is done implementation-wise, correct behavior should result.
11. I11. List of DOMASExceptions in the AS spec thus far: INVALID_CHARACTER_ERR, DUPLICATE_NAME_ERR, VALIDATION_ERR.
12. I12. Should names/namespaces of the various declarations be mutable during AS editing? AS ET agreed they should and are awaiting action by the XML CORE team.
13. I13. AS ET thinks the validate method and the error handler should be on Document, in CORE. If this doesn't happen, it needs to be on DocumentAS.
14. I14. RESOLVED: If entities are changed in the ASModel, the underlying model is unchanged until normalization.
15. I15. RESOLVED: Add option to control whether DOM CM is built from this document - solution is that the model is loaded (if there is one) and can be retrieved through the DocumentAS interface.
16. I16. RESOLVED: There is a way to add a new schema file to the existing active compound schema via setASModel().
17. I17. RESOLVED: Altering the document during error reporting, or mutation during validation terminates validation, and a warning will be produced if this happens.
18. I18. Proposal needed to rename the asHint, asLocation attributes and tie that into how to describe an `ASModel` [p.15] container of other ASModels.
19. I19. Proposal to revise getElementDeclaration method and introduce other methods on the DocumentAS interface.
20. I19. If implementation doesn't support AS-editing, need to have each set method throw an unsupported exception.

1.2. Abstract Schemas and AS-Editing Interfaces

A list of the proposed Abstract Schema data structures and functions follow, starting off with the data structures and "AS-editing" methods. Note that operations on the `ASModel` [p.15] that could result in its being invalid will be discovered during document validation and not during the AS editing operation, for example, removeNode(). Finally, note that an example element declaration: for `(A, (B* | C), D+)` can be described by the following:

```
ASElementDeclaration example = {
    strictMixedContent    = false;
    elementType           = STRING_DATATYPE;
    isPCDataOnly          = false;
    contentType           = ELEMENTS_CONTENTTYPE;
    tagname               = "example";
    ASContentModel        = exE;
    ASAttributeDecls      = null;
}

ASContentModel exE = {
    listOperator          = AS_SEQUENCE;
    minOccurs             = 1;
    maxOccurs             = 1;
    subModels             = {(ASElementDeclaration A),
                             (ASContentModel exBC),
                             (ASContentModel exD)};
}

ASElementDeclaration A = {
    strictMixedContent    = false;
    elementType           = STRING_DATATYPE;
    isPCDataOnly          = false;
    contentType           = ELEMENTS_CONTENTTYPE;
    tagname               = "A";
    ASContentModel        = null;
    ASAttributeDecls      = null;
}

ASContentModel exBC = {
    listOperator          = AS_CHOICE;
    minOccurs             = 1;
    maxOccurs             = 1;
    subModels             = {(ASContentModel exB),
                             (ASElementDeclaration C)};
}

ASContentModel exB = {
    listOperator          = AS_NONE;
    minOccurs             = 0;
    maxOccurs             = AS_UNBOUNDED;
    subModels             = {(ASElementDeclaration B)};
}
ASElementDeclaration B = {
    strictMixedContent    = false;
    elementType           = STRING_DATATYPE;
    isPCDataOnly          = false;
    contentType           = ELEMENTS_CONTENTTYPE;
    tagname               = "B";
    ASContentModel        = null;
    ASAttributeDecls      = null;
}

ASElementDeclaration C = {
    strictMixedContent    = false;
    elementType           = STRING_DATATYPE;
    isPCDataOnly          = false;
    contentType           = ELEMENTS_CONTENTTYPE;
    tagname               = "C";
    ASContentModel        = null;
    ASAttributeDecls      = null;
```

```
}

ASContentModel exD = {
    listOperator         = AS_NONE;
    minOccurs            = 1;
    maxOccurs            = AS_UNBOUNDED;
    subModels            = {(ASElementDeclaration D)};
}
ASElementDeclaration D = {
    strictMixedContent   = false;
    elementType          = STRING_DATATYPE;
    isPCDataOnly         = false;
    contentType          = ELEMENTS_CONTENTTYPE;
    tagname              = "D";
    ASContentModel       = null;
    ASAttributeDecls     = null;
}
```

Exception *DOMASException*

Abstract Schemas operations may throw a `DOMSystemException` as described in their descriptions.

IDL Definition

```
exception DOMASException {
  unsigned short   code;
};
// ASExceptionCode
const unsigned short      DUPLICATE_NAME_ERR              = 1;
const unsigned short      TYPE_ERR                        = 2;
const unsigned short      NO_AS_AVAILABLE                 = 3;
const unsigned short      WRONG_MIME_TYPE_ERR             = 4;
```

Definition group *ASExceptionCode*

An integer indicating the type of error generated.

Defined Constants

`DUPLICATE_NAME_ERR`
: If an element declaration already exists with the same name within an `AS_CHOICE` operator.

`NO_AS_AVAILABLE`
: If the `DocumentEditAS` [p.37] related to the node does not have any active `ASModel` [p.15] and `wfValidityCheckLevel` is set to `PARTIAL` or `STRICT_VALIDITY_CHECK`.

`TYPE_ERR`
: If the type of the `ASObject` [p.19] is neither an `ASContentModel` [p.30] nor an `ASElementDeclaration` [p.28] .

`WRONG_MIME_TYPE_ERR`
: When `mimeTypeCheck` is `true` and the input source has an incorrect MIME Type. See the attribute `mimeTypeCheck`.

Interface *ASModel*

To begin with, an abstract schema is a generic structure that could contain both internal and external subsets. An `ASModel` is an abstract object that could map to a DTD [XML], an XML Schema [XML Schema Part 0], a database schema, etc. An `ASModel` could represent either an internal or an external subset; hence an abstract schema could be composed of an `ASModel` representing the internal subset and an `ASModel` representing the external subset. Note that the `ASModel` representing the external subset could consult the `ASModel` representing the internal subset. Furthermore, the `ASModel` representing the internal subset could be set to null by the `setInternalAS` method as a mechanism for "removal". In addition, only one `ASModel` representing the external subset can be specified as "active" and it is possible that none are "active". Finally, the `ASModel` contains the factory methods needed to create a various types of ASObjects like `ASElementDeclaration` [p.28] , `ASAttributeDeclaration` [p.32] , etc.

IDL Definition

```
interface ASModel : ASObject {
  readonly attribute boolean          isNamespaceAware;
  readonly attribute unsigned short   usageLocation;
           attribute DOMString        asLocation;
           attribute DOMString        asHint;
  readonly attribute ASNamedObjectMap  elementDeclarations;
  readonly attribute ASNamedObjectMap  attributeDeclarations;
  readonly attribute ASNamedObjectMap  notationDeclarations;
  readonly attribute ASNamedObjectMap  entityDeclarations;
  readonly attribute ASNamedObjectMap  contentModelDeclarations;
  void               setASModel(in ASModel abstractSchema);
  ASObjectList       getASModels();
  void               removeAS(in ASModel as);
  boolean            validate();
  ASElementDeclaration createASElementDeclaration(in DOMString namespaceURI,
                                                  in DOMString name)
                                        raises(DOMException);
  ASAttributeDeclaration createASAttributeDeclaration(in DOMString namespaceURI,
                                                      in DOMString name)
                                        raises(DOMException);
  ASNotationDeclaration createASNotationDeclaration(in DOMString namespaceURI,
                                                    in DOMString name,
                                                    in DOMString systemId,
                                                    in DOMString publicId)
                                        raises(DOMException);
  ASEntityDeclaration createASEntityDeclaration(in DOMString name)
                                        raises(DOMException);
  ASContentModel     createASContentModel(in unsigned long minOccurs,
                                          in unsigned long maxOccurs,
                                          in unsigned short operator)
                                        raises(DOMASException);
};
```

Attributes

`asHint` of type `DOMString`
: The hint to locating an ASModel.

`asLocation` of type `DOMString`
: The URI reference.

`attributeDeclarations` of type `ASNamedObjectMap` [p.21] , readonly
: Instead of returning an all-in-one `ASObject` [p.19] with `ASModel` methods, have discernible top-level/"global" attribute declarations. If one attempts to add, set, or remove a node type other than the intended one, a hierarchy exception (or equivalent is thrown).

`contentModelDeclarations` of type `ASNamedObjectMap` [p.21] , readonly
: Instead of returning an all-in-one `ASObject` [p.19] with `ASModel` methods, have discernible top-level/"global content model declarations. If one attempts to add, set, or remove a node type other than the intended one, a hierarchy exception (or equivalent is thrown).

`elementDeclarations` of type `ASNamedObjectMap` [p.21] , readonly
: Instead of returning an all-in-one `ASObject` [p.19] with `ASModel` methods, have discernible top-level/"global" element declarations. If one attempts to add, set, or remove a node type other than the intended one, a hierarchy exception (or equivalent is thrown).

`entityDeclarations` of type `ASNamedObjectMap` [p.21] , readonly
: Instead of returning an all-in-one `ASObject` [p.19] with `ASModel` methods, have discernible top-level/"global" entity declarations. If one attempts to add, set, or remove a node type other than the intended one, a hierarchy exception (or equivalent is thrown).

`isNamespaceAware` of type `boolean`, readonly
: `true` if this `ASModel` defines the document structure in terms of namespaces and local names [XML Namespaces]; `false` if the document structure is defined only in terms of `QNames`.

`notationDeclarations` of type `ASNamedObjectMap` [p.21] , readonly
: Instead of returning an all-in-one `ASObject` [p.19] with `ASModel` methods, have discernible top-level/"global" notation declarations. If one attempts to add, set, or remove a node type other than the intended one, a hierarchy exception (or equivalent is thrown).

`usageLocation` of type `unsigned short`, readonly
: 0 if used internally, 1 if used externally, 2 if not all. An exception will be raised if it is incompatibly shared or in use as an internal subset.

Methods

`createASAttributeDeclaration`
: Creates an attribute declaration.

Parameters

`namespaceURI` of type `DOMString`
: The *namespace URI* [p.135] of the attribute being declared.

`name` of type `DOMString`
: The name of the attribute. The format of the name could be an NCName as defined by XML Namespaces or a Name as defined by XML 1.0; it’s ASModel-dependent.

Return Value

`ASAttributeDeclaration` [p.32]	A new `ASAttributeDeclaration` object with appropriate attributes set by input parameters.

Exceptions

`DOMException`	INVALID_CHARACTER_ERR: Raised if the input `name` parameter contains an illegal character.

`createASContentModel`

Creates an object which describes part of an `ASElementDeclaration` [p.28] 's content model.

Parameters

`minOccurs` of type `unsigned long`
: The minimum occurrence for the subModels of this `ASContentModel` [p.30] .

`maxOccurs` of type `unsigned long`
: The maximum occurrence for the subModels of this `ASContentModel` [p.30] .

`operator` of type `unsigned short`
: operator of type `AS_CHOICE`, `AS_SEQUENCE`, `AS_ALL` or `AS_NONE`.

Return Value

`ASContentModel` [p.30]	A new `ASContentModel` object.

Exceptions

`DOMASException` [p.14]	A DOMASException, e.g., `minOccurs > maxOccurs`.

`createASElementDeclaration`

Creates an *element* [p.135] declaration for the element type specified.

Parameters

`namespaceURI` of type `DOMString`
: The `namespace URI` of the element type being declared.

`name` of type `DOMString`
: The name of the element. The format of the name could be an NCName as defined by XML Namespaces or a Name as defined by XML 1.0; it's ASModel-dependent.

Return Value

`ASElementDeclaration` [p.28]	A new `ASElementDeclaration` object with `name` attribute set to `tagname` and `namespaceURI` set to `systemId`. Other attributes of the element declaration are set through `ASElementDeclaration` interface methods.

Exceptions

`DOMException`	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createASEntityDeclaration`

Creates an ASEntityDeclaration.

Parameters

`name` of type `DOMString`

The name of the entity being declared.

Return Value

`ASEntityDeclaration` [p.33]	A new `ASEntityDeclaration` object with `entityName` attribute set to name.

Exceptions

`DOMException`	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createASNotationDeclaration`

Creates a new notation declaration.

Parameters

`namespaceURI` of type `DOMString`

The *namespace URI* [p.135] of the notation being declared.

`name` of type `DOMString`

The name of the notation. The format of the name could be an NCName as defined by XML Namespaces or a Name as defined by XML 1.0; it’s ASModel-dependent.

`systemId` of type `DOMString`

The system identifier for the notation declaration.

`publicId` of type `DOMString`

The public identifier for the notation declaration.

Return Value

`ASNotationDeclaration` [p.34]	A new `ASNotationDeclaration` object with `notationName` attribute set to `name` and `publicId` and `systemId` set to the corresponding fields.

Exceptions

`DOMException`	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`getASModels`

To retrieve a list of nested ASModels without reference to names.

Return Value

`ASObjectList` [p.21]	A list of ASModels.

No Parameters

No Exceptions

`removeAS`

Removes only the specified `ASModel` from the list of `ASModel`s.

Parameters

`as` of type `ASModel` [p.15]

AS to be removed.

No Return Value

No Exceptions

`setASModel`

This method will allow the nesting or "importation" of ASModels.

Parameters

`abstractSchema` of type `ASModel` [p.15]

ASModel to be set. Subsequent calls will nest the ASModels within the specified `ownerASModel`.

No Return Value

No Exceptions

`validate`

Determines if an `ASModel` itself is valid, i.e., confirming that it’s well-formed and valid per its own formal grammar.

Return Value

`boolean`	`true` if the `ASModel` is valid, `false` otherwise.

No Parameters

No Exceptions

Interface *ASObject*

The `ASObject` interface is analogous to a `Node` in [DOM Level 3 Core], e.g., an element declaration.

Opaque.

IDL Definition

```
interface ASObject {

  // ASObjectType
  const unsigned short      AS_ELEMENT_DECLARATION         = 1;
  const unsigned short      AS_ATTRIBUTE_DECLARATION       = 2;
  const unsigned short      AS_NOTATION_DECLARATION        = 3;
  const unsigned short      AS_ENTITY_DECLARATION          = 4;
  const unsigned short      AS_CONTENTMODEL                = 5;
  const unsigned short      AS_MODEL                       = 6;

  readonly attribute unsigned short   asNodeType;
           attribute ASModel          ownerASModel;
           attribute DOMString        nodeName;
           attribute DOMString        prefix;
           attribute DOMString        localName;
           attribute DOMString        namespaceURI;
  ASObject           cloneASObject(in boolean deep);
};
```

Definition group *ASObjectType*

An integer indicating which type of `ASObject` this is.

Defined Constants

`AS_ATTRIBUTE_DECLARATION`
: The node is an `ASAttributeDeclaration` [p.32] .

`AS_CONTENTMODEL`
: The node is a `ASContentModel` [p.30] .

`AS_ELEMENT_DECLARATION`
: The node is an `ASElementDeclaration` [p.28] .

`AS_ENTITY_DECLARATION`
: The node is an `ASEntityDeclaration` [p.33] .

`AS_MODEL`
: The node is a `ASModel` [p.15] .

`AS_NOTATION_DECLARATION`
: The node is a `ASNotationDeclaration` [p.34] .

Attributes

`asNodeType` of type `unsigned short`, readonly
: A code representing the underlying object as defined above.

`localName` of type `DOMString`
: Returns the local part of the *qualified name* [p.136] of this `ASObject`.

`namespaceURI` of type `DOMString`
: The *namespace URI* [p.135] of this node, or `null` if it is unspecified. [XML Schema Part 1] defines how a *namespace URI* [p.135] is attached to schema components.

`nodeName` of type `DOMString`
: The `name` of this `ASObject` depending on the `ASObject` type.

`ownerASModel` of type `ASModel` [p.15]
: The `ASModel` [p.15] object associated with this `ASObject`. For a node of type `AS_MODEL`, this is `null`.

`prefix` of type `DOMString`
: The *namespace prefix* [p.135] of this node, or `null` if it is unspecified.

Methods

`cloneASObject`
: Creates a copy of this `ASObject`. See text for `cloneNode` off of `Node` but substitute AS functionality.

Parameters

`deep` of type `boolean`

Setting the `deep` flag on, causes the whole subtree to be duplicated. Setting it to `false` only duplicates its immediate child nodes.

Return Value

`ASObject` [p.19]	Cloned `ASObject`.

No Exceptions

Interface *ASObjectList*

The `ASObjectList` interface provides the abstraction of an ordered collection of AS nodes, without defining or constraining how this collection is implemented. `ASObjectList` objects in the DOM AS are *live* [p.135] .

IDL Definition

```
interface ASObjectList {
  readonly attribute unsigned long    length;
  ASObject           item(in unsigned long index);
};
```

Attributes

`length` of type `unsigned long`, readonly

The number of `ASObjects` [p.19] in the list. The range of valid *child* [p.135] node indices is 0 to `length-1` inclusive.

Methods

`item`

Returns the `index`th item in the collection. The index starts at 0. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

Parameters

`index` of type `unsigned long`

index into the collection.

Return Value

`ASObject` [p.19]	The `ASObject` at the `index`th position in the `ASObjectList`, or `null` if that is not a valid index.

No Exceptions

Interface *ASNamedObjectMap*

Objects implementing the `ASNamedObjectMap` interface are used to represent collections of abstract schema nodes that can be accessed by name. Note that `ASNamedObjectMap` does not inherit from `ASObjectList` [p.21] ; `ASNamedObjectMaps` are not maintained in any particular order. Objects contained in an object implementing `ASNamedObjectMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `ASNamedObjectMap`, and does not imply that the DOM specifies an order to these `ASObjects` [p.19] .

`ASNamedObjectMap` object in the DOM are *live* [p.135] .

IDL Definition

```
interface ASNamedObjectMap {
  readonly attribute unsigned long    length;
  ASObject           getNamedItem(in DOMString name);
  ASObject           getNamedItemNS(in DOMString namespaceURI,
                                    in DOMString localName);
  ASObject           item(in unsigned long index);
  ASObject           removeNamedItem(in DOMString name)
                                        raises(DOMException);
  ASObject           removeNamedItemNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                        raises(DOMException);
  ASObject           setNamedItem(in ASObject newASObject)
                                        raises(DOMException);
  ASObject           setNamedItemNS(in ASObject newASObject)
                                        raises(DOMException);
};
```

Attributes

`length` of type `unsigned long`, readonly

The number of `ASObjects` [p.19] in the `ASObjectList` [p.21] . The range of valid *child* [p.135] node indices is 0 to `length-1` inclusive.

Methods

`getNamedItem`

Retrieves an `ASObject` [p.19] specified by name.

Parameters

`name` of type `DOMString`

The `nodeName` of an `ASObject` [p.19] to retrieve.

Return Value

`ASObject` [p.19]	An `ASObject` with specified node name and `null` if the map does not contain an *element* [p.135] with the given name.

No Exceptions

`getNamedItemNS`

Retrieves an `ASObject` [p.19] specified by *local name* [p.135] and *namespace URI* [p.135] .

Parameters

`namespaceURI` of type `DOMString`

The *namespace URI* [p.135] of the `ASObject` [p.19] to retrieve.

`localName` of type `DOMString`

The *local name* [p.135] of the `ASObject` [p.19] to retrieve.

Return Value

`ASObject` [p.19]	A `ASObject` (of any type) with the specified local name and namespace URI, or `null` if they do not identify any `ASObject` in this map.

No Exceptions

`item`

Returns the `index`th item in the map. The index starts at `0`. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

Parameters

`index` of type `unsigned long`

The position in the map from which the item is to be retrieved.

Return Value

`ASObject` [p.19]	The `ASObject` at the `index`th position in the `ASNamedObjectMap`, or `null` if that is not a valid index.

No Exceptions

`removeNamedItem`

Removes an `ASObject` [p.19] specified by a `nodeName`.

Parameters

`name` of type `DOMString`

The `nodeName` of the `ASObject` [p.19] to be removed.

Return Value

`ASObject` [p.19]	The `ASObject` removed from this map if an `ASObject` with such a name exists.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if there is no node named `name` in this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`removeNamedItemNS`

Removes an `ASObject` [p.19] specified by a namespace URI and a local name.

Parameters

`namespaceURI` of type `DOMString`

The *namespace URI* [p.135] of the `ASObject` [p.19] to be removed.

`localName` of type `DOMString`

The *local name* [p.135] of the `ASObject` [p.19] to remove.

Return Value

`ASObject` [p.19]	The `ASObject` removed from this map if an `ASObject` with such a local name and namespace URI exists.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if there is no node with the specified `namespaceURI` and `localName` in this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`setNamedItem`

Adds an `ASObject` [p.19] using its `nodeName` attribute. If an `ASObject` with that name is already present in this map, it is replaced by the new one.

Parameters

`newASObject` of type `ASObject` [p.19]

The `ASObject` to be inserted in the map with its `nodeName` as the key.

Return Value

`ASObject` [p.19]	If the new node replaces an existing one, the replaced node is returned, otherwise `null`.

Exceptions

`DOMException`	WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different `ASModel` [p.15] than the one that created this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly. HIERARCHY_REQUEST_ERR: Raised if an attempt is made to add a node doesn’t belong in this `ASNamedObjectMap`.

`setNamedItemNS`

Adds an `ASObject` [p.19] using its `namespaceURI` and `localName`. If an `ASObject` with the same `namespaceURI` and `localName` is already present in this map, it is replaced by the new one.

Parameters

`newASObject` of type `ASObject` [p.19]

The `ASObject` to be inserted in the map.The `ASObject` will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Return Value

`ASObject` [p.19]	If the new node replaces an existing one, the replaced node is returned, otherwise `null`.

Exceptions

`DOMException`	`WRONG_DOCUMENT_ERR`: Raised if `arg` was created from a different `ASModel` [p.15] than the one that created this map. `NO_MODIFICATION_ALLOWED_ERR`: Raised if this map is readonly. `HIERARCHY_REQUEST_ERR`: Raised if an attempt is made to add a node doesn't belong in this `ASNamedObjectMap`.

Interface *ASDataType*

The datatypes supported by DOM AS implementations. Further datatypes may be added in the Schema/PSVI spec.

IDL Definition

```
interface ASDataType {
  readonly attribute unsigned short   dataType;

  // DATA_TYPES
  const unsigned short      STRING_DATATYPE                = 1;
  const unsigned short      NOTATION_DATATYPE              = 10;
  const unsigned short      ID_DATATYPE                    = 11;
  const unsigned short      IDREF_DATATYPE                 = 12;
  const unsigned short      IDREFS_DATATYPE                = 13;
  const unsigned short      ENTITY_DATATYPE                = 14;
  const unsigned short      ENTITIES_DATATYPE              = 15;
  const unsigned short      NMTOKEN_DATATYPE               = 16;
  const unsigned short      NMTOKENS_DATATYPE              = 17;
  const unsigned short      BOOLEAN_DATATYPE               = 100;
  const unsigned short      FLOAT_DATATYPE                 = 101;
  const unsigned short      DOUBLE_DATATYPE                = 102;
  const unsigned short      DECIMAL_DATATYPE               = 103;
  const unsigned short      HEXBINARY_DATATYPE             = 104;
  const unsigned short      BASE64BINARY_DATATYPE          = 105;
  const unsigned short      ANYURI_DATATYPE                = 106;
  const unsigned short      QNAME_DATATYPE                 = 107;
  const unsigned short      DURATION_DATATYPE              = 108;
  const unsigned short      DATETIME_DATATYPE              = 109;
  const unsigned short      DATE_DATATYPE                  = 110;
  const unsigned short      TIME_DATATYPE                  = 111;
  const unsigned short      GYEARMONTH_DATATYPE            = 112;
  const unsigned short      GYEAR_DATATYPE                 = 113;
  const unsigned short      GMONTHDAY_DATATYPE             = 114;
  const unsigned short      GDAY_DATATYPE                  = 115;
  const unsigned short      GMONTH_DATATYPE                = 116;
  const unsigned short      INTEGER                        = 117;
  const unsigned short      NAME_DATATYPE                    = 200;
  const unsigned short      NCNAME_DATATYPE                  = 201;
  const unsigned short      NORMALIZEDSTRING_DATATYPE        = 202;
  const unsigned short      TOKEN_DATATYPE                   = 203;
  const unsigned short      LANGUAGE_DATATYPE                = 204;
  const unsigned short      NONPOSITIVEINTEGER_DATATYPE      = 205;
  const unsigned short      NEGATIVEINTEGER_DATATYPE         = 206;
  const unsigned short      LONG_DATATYPE                    = 207;
  const unsigned short      INT_DATATYPE                     = 208;
  const unsigned short      SHORT_DATATYPE                   = 209;
  const unsigned short      BYTE_DATATYPE                    = 210;
  const unsigned short      NONNEGATIVEINTEGER_DATATYPE      = 211;
  const unsigned short      UNSIGNEDLONG_DATATYPE            = 212;
  const unsigned short      UNSIGNEDINT_DATATYPE             = 213;
  const unsigned short      UNSIGNEDSHORT_DATATYPE           = 214;
  const unsigned short      UNSIGNEDBYTE_DATATYPE            = 215;
  const unsigned short      POSITIVEINTEGER_DATATYPE         = 216;
  const unsigned short      OTHER_SIMPLE_DATATYPE            = 1000;
  const unsigned short      COMPLEX_DATATYPE                 = 1001;
};
```

Definition group *DATA_TYPES*

An integer indicating which datatype this is.

Defined Constants

`ANYURI_DATATYPE`
: Then *uri reference* data type as defined in [XML Schema Part 2].

`BASE64BINARY_DATATYPE`
: The *base64binary* data type as defined in [XML Schema Part 2].

`BOOLEAN_DATATYPE`
: A code representing the *boolean* data type as defined in [XML Schema Part 2].

`BYTE_DATATYPE`
: The *byte* data type as defined in [XML Schema Part 2].

`COMPLEX_DATATYPE`
: The user-defined complex data type as defined in [XML Schema Part 2].

`DATETIME_DATATYPE`
: The *datetime* data type as defined in [XML Schema Part 2].

`DATE_DATATYPE`
: The *date* data type as defined in [XML Schema Part 2].

`DECIMAL_DATATYPE`
: The *decimal* data type as defined in [XML Schema Part 2].

`DOUBLE_DATATYPE`
: A code representing the *double* data type as defined in [XML Schema Part 2].

`DURATION_DATATYPE`
: The *duration* data type as defined in [XML Schema Part 2].

`ENTITIES_DATATYPE`
: The *ENTITIES* data type as defined in [XML Schema Part 2].

`ENTITY_DATATYPE`
: The *ENTITY* data type as defined in [XML Schema Part 2].

`FLOAT_DATATYPE`
: A code representing the *float* data type as defined in [XML Schema Part 2].

`GDAY_DATATYPE`
: The *day* data type as defined in [XML Schema Part 2].

`GMONTHDAY_DATATYPE`
: The *monthday* data type as defined in [XML Schema Part 2].

`GMONTH_DATATYPE`
: The *month* data type as defined in [XML Schema Part 2].

`GYEARMONTH_DATATYPE`
: The *yearmonth* data type as defined in [XML Schema Part 2].

`GYEAR_DATATYPE`
: The *year* data type as defined in [XML Schema Part 2].

`HEXBINARY_DATATYPE`
: The *hexbinary* data type as defined in [XML Schema Part 2].

`IDREFS_DATATYPE`
: The *IDREFS* data type as defined in [XML Schema Part 2].

`IDREF_DATATYPE`
: The *IDREF* data type as defined in [XML Schema Part 2].

`ID_DATATYPE`
: The *ID* data type as defined in [XML Schema Part 2].

`INTEGER`
: The *integer* data type as defined in [XML Schema Part 2].

`INT_DATATYPE`
: The *integer* data type as defined in [XML Schema Part 2].

`LANGUAGE_DATATYPE`
: The *Language* data type as defined in [XML Schema Part 2].

`LONG_DATATYPE`
: Then *long* data type as defined in [XML Schema Part 2].

`NAME_DATATYPE`
: A code representing the *Name* data type as defined in [XML Schema Part 2].

`NCNAME_DATATYPE`
: A code representing the *NCName* data type as defined in [XML Schema Part 2].

`NEGATIVEINTEGER_DATATYPE`
: Then *negative integer* data type as defined in [XML Schema Part 2].

`NMTOKENS_DATATYPE`
: The *NMTOKENS* data type as defined in [XML Schema Part 2].

`NMTOKEN_DATATYPE`
: The *NMTOKEN* data type as defined in [XML Schema Part 2].

`NONNEGATIVEINTEGER_DATATYPE`
: The *non-negative integer* data type as defined in [XML Schema Part 2].

`NONPOSITIVEINTEGER_DATATYPE`
: The *Non-positive integer* data type as defined in [XML Schema Part 2].

`NORMALIZEDSTRING_DATATYPE`
: A code representing the *Normalized string* data type as defined in [XML Schema Part 2].

`NOTATION_DATATYPE`
The *NOTATION* data type as defined in [XML Schema Part 2].

`OTHER_SIMPLE_DATATYPE`
The other simple data type as defined in [XML Schema Part 2].

`POSITIVEINTEGER_DATATYPE`
The *positive integer* data type as defined in [XML Schema Part 2].

`QNAME_DATATYPE`
Then *XML qualified name* data type as defined in [XML Schema Part 2].

`SHORT_DATATYPE`
The *short* data type as defined in [XML Schema Part 2].

`STRING_DATATYPE`
A code representing the *string* data type as defined in [XML Schema Part 2].

`TIME_DATATYPE`
The *time* data type as defined in [XML Schema Part 2].

`TOKEN_DATATYPE`
The *token* data type as defined in [XML Schema Part 2].

`UNSIGNEDBYTE_DATATYPE`
The *unsigned byte* data type as defined in [XML Schema Part 2].

`UNSIGNEDINT_DATATYPE`
The *unsigned integer* data type as defined in [XML Schema Part 2].

`UNSIGNEDLONG_DATATYPE`
The *unsigned long* data type as defined in [XML Schema Part 2].

`UNSIGNEDSHORT_DATATYPE`
The *unsigned short* data type as defined in [XML Schema Part 2].

Attributes

`dataType` of type `unsigned short`, readonly
One of the enumerated codes representing the data type.

Interface *ASElementDeclaration*

The element name along with the content specification in the context of an `ASObject` [p.19] .

IDL Definition

```
interface ASElementDeclaration : ASObject {

  // CONTENT_MODEL_TYPES
  const unsigned short      EMPTY_CONTENTTYPE              = 1;
  const unsigned short      ANY_CONTENTTYPE                = 2;
  const unsigned short      MIXED_CONTENTTYPE              = 3;
  const unsigned short      ELEMENTS_CONTENTTYPE           = 4;

           attribute boolean          strictMixedContent;
           attribute ASDataType       elementType;
           attribute boolean          isPCDataOnly;
           attribute unsigned short   contentType;
           attribute DOMString        systemId;
           attribute ASContentModel   asCM;
           attribute ASNamedObjectMap  ASAttributeDecls;
  void              addASAttributeDecl(in ASAttributeDeclaration attributeDecl);
  ASAttributeDeclaration removeASAttributeDecl(in ASAttributeDeclaration attributeDecl);
};
```

Definition group *CONTENT_MODEL_TYPES*

Defined Constants

`ANY_CONTENTTYPE`
Represents an ANY content type for an Element declaration.

`ELEMENTS_CONTENTTYPE`
Represents an ELEMENTS only content type for an Element declaration.

`EMPTY_CONTENTTYPE`
Represents an EMPTY content type for an Element declaration.

`MIXED_CONTENTTYPE`
Represents a MIXED content type for an Element declaration. Note that `isPCDataOnly` would also need to checked, in addition to this, if an element's content model was simply text, as an example.

Attributes

`ASAttributeDecls` of type `ASNamedObjectMap` [p.21]
The`ASNamedObjectMap` [p.21] containing `ASAttributeDeclarations` [p.32] for all the attributes that can appear on this type of element.

`asCM` of type `ASContentModel` [p.30]
The *content model* [p.135] of element.

`contentType` of type `unsigned short`
The content type of the element. One of `EMPTY_CONTENTTYPE`, `ANY_CONTENTTYPE`, `MIXED_CONTENTTYPE`, `ELEMENTS_CONTENTTYPE`.

`elementType` of type `ASDataType` [p.25]
Datatype of the element.

`isPCDataOnly` of type `boolean`
Boolean defining whether the element type contains child elements and PCDATA or PCDATA only for mixed element types. `true` if the element is of type PCDATA only. Relevant only for mixed content type elements.

`strictMixedContent` of type `boolean`
A boolean defining whether the element order and number of the *child* [p.135] elements for mixed content type has to be respected or not. For example XML Schema defined mixed content types the order is important and needs to be respected whether for DTD based AS the order and number of *child* [p.135] elements are not important.

`systemId` of type `DOMString`
the URI reference representing the system identifier for the notation declaration, if present, `null` otherwise.

Methods

`addASAttributeDecl`
Adds an `ASAttributeDeclaration` [p.32] for the element being declared.

Parameters

`attributeDecl` of type `ASAttributeDeclaration` [p.32]
The new attribute to add. If the attribute declaration already exists for the element, the call does not have any effect.

No Return Value

No Exceptions

`removeASAttributeDecl`
Removes an `ASAttributeDeclaration` [p.32] from the element being declared.

Parameters

`attributeDecl` of type `ASAttributeDeclaration` [p.32]
: The attribute declaraition to be removed. If the attribute declaration does not exist for the element, the call does not have any effect.

Return Value

`ASAttributeDeclaration` [p.32]	`null` if the attribute does not exist. Otherwise returns the attribute being removed.

No Exceptions

Interface *ASContentModel*

The content model of a declared element.

IDL Definition

```
interface ASContentModel : ASObject {
  const unsigned long       AS_UNBOUNDED                  = MAX_VALUE;

  // ASContentModelType
  const unsigned short      AS_SEQUENCE                   = 0;
  const unsigned short      AS_CHOICE                     = 1;
  const unsigned short      AS_ALL                        = 2;
  const unsigned short      AS_NONE                       = 3;

           attribute unsigned short   listOperator;
           attribute unsigned long    minOccurs;
           attribute unsigned long    maxOccurs;
           attribute ASObjectList     subModels;
  void               removesubModel(in ASObject oldNode);
  void               insertsubModel(in ASObject newNode)
                                        raises(DOMASException);
  unsigned long      appendsubModel(in ASObject newNode)
                                        raises(DOMASException);
};
```

Constant *AS_UNBOUNDED*

Signifies unbounded upper limit. The MAX_VALUE value is `0xFFFFFFFF FFFFFFFF`. (***ED:*** This needs to be better defined in the generated bindings.)

Definition group *ASContentModelType*

An integer indicating which type of `ASContentModel` this is.

Defined Constants

`AS_ALL`
: All of the above.

`AS_CHOICE`
: This constant value signifies a choice operator. For example, in a DTD, this would be the '`|`' operator.

`AS_NONE`
: None of the above, i.e., neither a choice nor sequence operator.

`AS_SEQUENCE`
: This constant value signifies a sequence operator. For example, in a DTD, this would be the '`,`' operator.

Attributes

`listOperator` of type `unsigned short`
: One of `AS_CHOICE`, `AS_SEQUENCE`, `AS_ALL` or `AS_NONE`. The operator is applied to all the components(ASObjects) in the `subModels`. For example, if the list operator is `AS_CHOICE` and the components in subModels are a, b and c then the abstract schema for the element being declared is `(a|b|c)`.

`maxOccurs` of type `unsigned long`
: maximum occurrence for this content particle. Its value may be `0`, a positive integer, or `AS_UNBOUNDED` to indicate that no upper limit has been set.

`minOccurs` of type `unsigned long`
: min occurrence for this content particle. Its value may be 0 or a positive integer.

`subModels` of type `ASObjectList` [p.21]
: Pointers to `ASObject` [p.19] s such as `ASElementDeclaration`s and further `ASContentModel`s.

Methods

`appendsubModel`
: Appends a new node to the end of the list representing the`subModels`.

Parameters

`newNode` of type `ASObject` [p.19]
: The new node to be appended.

Return Value

`unsigned long`	the length of the `subModels`.

Exceptions

`DOMASException` [p.14]	`DUPLICATE_NAME_ERR`: Raised if a element declaration already exists with the same name within an `AS_CHOICE` operator. `TYPE_ERR`: Raised if type is neither an `ASContentModel` nor an `ASElementDeclaration` [p.28] .

`insertsubModel`
: Inserts a new node in the submodel. Nodes that already exist in the list are moved as needed.

Parameters

`newNode` of type `ASObject` [p.19]
: The new node to be inserted.

Exceptions

`DOMASException` [p.14]	`DUPLICATE_NAME_ERR`: Raised if a element declaration already exists with the same name within an `AS_CHOICE` operator.

No Return Value

`removesubModel`

Removes the `ASObject` [p.19] in the submodel. Nodes that already exist in the list are moved as needed.

Parameters

`oldNode` of type `ASObject` [p.19]

The node to be removed.

No Return Value

No Exceptions

Interface *ASAttributeDeclaration*

An attribute declaration in the context of a `ASObject` [p.19] .

Issue ASAttributeDeclaration-1:

The constant 'REQUIRED' is missing from this interface.

IDL Definition

```
interface ASAttributeDeclaration : ASObject {

  // VALUE_TYPES
  const unsigned short      VALUE_NONE                    = 0;
  const unsigned short      VALUE_DEFAULT                 = 1;
  const unsigned short      VALUE_FIXED                   = 2;

           attribute ASDataType       dataType;
           attribute DOMString        dataValue;
           attribute DOMString        enumAttr;
           attribute ASObjectList     ownerElements;
           attribute unsigned short   defaultType;
};
```

Definition group *VALUE_TYPES*

Defined Constants

`VALUE_DEFAULT`

Indicates that the there is a default value constraint.

`VALUE_FIXED`

Indicates that there is a fixed value constraint for this attribute.

`VALUE_NONE`

Describes that the attribute does not have any value constraint.

Attributes

`dataType` of type `ASDataType` [p.25]

Datatype of the attribute.

`dataValue` of type `DOMString`

Default or fixed value.

`defaultType` of type `unsigned short`
: Constraint type if any for this attribute.

`enumAttr` of type `DOMString`
: Valid attribute values, separated by commas, in a string.

`ownerElements` of type `ASObjectList` [p.21]
: Owner elements `ASObject` [p.19] of attribute, meaning that an attribute declaration can be shared by multiple elements.

Interface *ASEntityDeclaration*

Models a general entity declaration in an abstract schema.

(***ED:*** The abstract schema does not handle any parameter entity. It is assumed that the parameter entities are expanded by the implementation as the abstract schema is built.)

IDL Definition

```
interface ASEntityDeclaration : ASObject {

  // EntityType
  const unsigned short      INTERNAL_ENTITY                = 1;
  const unsigned short      EXTERNAL_ENTITY                = 2;

           attribute unsigned short   entityType;
           attribute DOMString        entityValue;
           attribute DOMString        systemId;
           attribute DOMString        publicId;
};
```

Definition group *EntityType*

An integer indicating which type of entity this is.

Defined Constants

`EXTERNAL_ENTITY`
: constant defining an external entity.

`INTERNAL_ENTITY`
: constant defining an internal entity.

Attributes

`entityType` of type `unsigned short`
: The type of the entity as defined above.

`entityValue` of type `DOMString`
: The replacement text for the internal entity. The entity references within the replacement text are kept intact. For an entity of type `EXTERNAL_ENTITY`, this is `null`.

`publicId` of type `DOMString`
: The string representing the public identifier for this notation declaration, if present; `null` otherwise.

`systemId` of type `DOMString`
: the URI reference representing the system identifier for the notation declaration, if present, `null` otherwise.

Interface *ASNotationDeclaration*

This interface represents a notation declaration.

IDL Definition

```
interface ASNotationDeclaration : ASObject {
           attribute DOMString        systemId;
           attribute DOMString        publicId;
};
```

Attributes

`publicId` of type `DOMString`
: The string representing the public identifier for this notation declaration, if present; `null` otherwise.

`systemId` of type `DOMString`
: the URI reference representing the system identifier for the notation declaration, if present, `null` otherwise.

1.3. Validation and Other Interfaces

This section contains "Validation and Other" methods common to both the document-editing and AS-editing worlds (includes `DOMImplementation` methods).

Interface *DocumentAS*

This interface extends the `Document` interface with additional methods for both document and AS editing.

IDL Definition

```
interface DocumentAS {
           attribute ASModel          activeASModel;
           attribute ASObjectList     boundASModels;
  ASModel            getInternalAS();
  void               setInternalAS(in ASModel as);
  void               addAS(in ASModel as);
  void               removeAS(in ASModel as);
  ASElementDeclaration getElementDeclaration()
                                        raises(DOMException);
  void               validate()
                                        raises(DOMASException);
};
```

Attributes

`activeASModel` of type `ASModel` [p.15]
: The active external ASModel. Note that the active external `ASModel` [p.15] is responsible for consulting the internal ASModel, so if an attribute is declared in the internal `ASModel` and the corresponding `ownerElements` points to a `ASElementDeclaration` [p.28] s defined in the active external ASModel, changing the active external ASModel will cause the `ownerElements` to be recomputed. If the `ownerElements` is not defined in the

newly active external ASModel, the `ownerElements` will be an empty node list.

`boundASModels` of type `ASObjectList` [p.21]

A list of `ASObject` [p.19] s of type `AS_MODEL`s associated with a document. The `addAS` method associates a `ASModel` [p.15] with a document.

Methods

`addAS`

Associate a `ASModel` [p.15] with a document. Can be invoked multiple times to result in a list of `ASModel`s. Note that only one internal `ASModel` is associated with the document, however, and that only one of the possible list of `ASModel`s is active at any one time.

Parameters

`as` of type `ASModel` [p.15]

`ASModel` to be associated with the document.

No Return Value

No Exceptions

`getElementDeclaration`

Gets the AS editing object describing this element

Issue getElementDeclaration-1:

This method needs to be changed and others added.

Return Value

`ASElementDeclaration` [p.28]	ASElementDeclaration object if the implementation supports "`AS-EDIT`" feature. Otherwise `null`.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if no `ASModel` [p.15] is present.

No Parameters

`getInternalAS`

Retrieve the internal `ASModel` [p.15] of a document.

Return Value

`ASModel` [p.15]	`ASModel`.

No Parameters

No Exceptions

`removeAS`

Removes a `ASModel` [p.15] associated with a document. Can be invoked multiple times to remove a number of these in the list of `ASModel`s.

Parameters

`as` of type `ASModel` [p.15]

The `ASModel` to be removed.

No Return Value

No Exceptions

`setInternalAS`

Sets the internal subset `ASModel` [p.15] of a document. This could be null as a mechanism for "removal".

Parameters

`as` of type `ASModel` [p.15]

`ASModel` to be the internal subset of the document.

No Return Value

No Exceptions

`validate`

Validates the document against the `ASModel` [p.15] .

Exceptions

`DOMASException` [p.14]	

No Parameters

No Return Value

Interface *DOMImplementationAS*

This interface allows creation of an `ASModel` [p.15] . The expectation is that an instance of the `DOMImplementationAS` interface can be obtained by using binding-specific casting methods on an instance of the `DOMImplementation` interface when the DOM implementation supports the feature "`AS-EDIT`".

IDL Definition

```
interface DOMImplementationAS {
  ASModel            createAS(in boolean isNamespaceAware);
  DOMASBuilder       createDOMASBuilder();
  DOMASWriter        createDOMASWriter();
};
```

Methods

`createAS`

Creates an ASModel.

Parameters

`isNamespaceAware` of type `boolean`

Allow creation of `ASModel` [p.15] with this attribute set to a specific value.

Return Value

`ASModel` [p.15]	A `null` return indicates failure. Issue createAS-1: what is a failure? Could be a system error.

No Exceptions

`createDOMASBuilder`
Creates an `DOMASBuilder` [p.49] .
Issue createDOMASBuilder-1:
Do we need the method since we already have `DOMImplementationLS.createDOMBuilder` [p.65] ?
Return Value

`DOMASBuilder` [p.49]

No Parameters
No Exceptions

`createDOMASWriter`
Creates an `DOMASWriter` [p.51] .
Return Value

`DOMASWriter` [p.51]

No Parameters
No Exceptions

1.4. Document-Editing Interfaces

This section contains "Document-editing" methods (includes `Node`, `Element`, `Text` and `Document` methods).

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "AS-DOC" and "3.0" (respectively) to determine whether or not the Document-Editing interfaces of the Abstract Schemas module are supported by the implementation.

Interface *DocumentEditAS*

This interface extends the `NodeEditAS` [p.38] interface with additional methods for both document and AS editing.

IDL Definition

```
interface DocumentEditAS : NodeEditAS {
           attribute boolean          continuousValidityChecking;
};
```

Attributes
`continuousValidityChecking` of type `boolean`
An attribute specifying whether continuous checking for the validity of the document is enforced or not. Setting this to `true` will result in an exception being thrown, i.e., `VALIDATION_ERR`, for documents that are invalid at the time of the call. If the document is invalid, then this attribute will remain `false`. This attribute is `false` by default.

(***ED:*** Add VALIDATION_ERR code to the list of constants in DOMASException.)

Interface *NodeEditAS*

This interface extends a `Node` from [DOM Level 3 Core] with additional methods for guided document editing. The expectation is that an instance of the `DOMImplementationAS` [p.36] interface can be obtained by using binding-specific casting methods on an instance of the `DOMImplementation` interface when the DOM implementation supports the feature "`AS-DOC`".

IDL Definition

```
interface NodeEditAS {

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

  boolean            canInsertBefore(in Node newChild, 
                                     in Node refChild);
  boolean            canRemoveChild(in Node oldChild);
  boolean            canReplaceChild(in Node newChild, 
                                     in Node oldChild);
  boolean            canAppendChild(in Node newChild);
  boolean            isNodeValid(in boolean deep, 
                                 in unsigned short wFValidityCheckLevel)
                                        raises(DOMASException);
};
```

Definition group *ASCheckType*

An integer indicating which type of validation this is.

Defined Constants

`NS_WF_CHECK`
Check for namespace well-formedness includes `WF_CHECK`.

`PARTIAL_VALIDITY_CHECK`
Checks for whether this node is *partially valid* [p.135] . It includes `NS_WF_CHECK`.

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

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

Methods

`canAppendChild`
Has the same arguments as `AppendChild`.

Parameters

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

Return Value

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

No Exceptions

`canInsertBefore`

Determines whether the `insertBefore` operation from the `Node` interface would make this document invalid with respect to the currently active AS.

Issue canInsertBefore-1:

Describe "valid" when referring to partially completed documents.

Parameters

`newChild` of type `Node`

`Node` to be inserted.

`refChild` of type `Node`

Reference `Node`.

Return Value

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

No Exceptions

`canRemoveChild`

Has the same arguments as `RemoveChild`.

Parameters

`oldChild` of type `Node`

`Node` to be removed.

Return Value

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

No Exceptions

`canReplaceChild`

Has the same arguments as `ReplaceChild`.

Parameters

`newChild` of type `Node`

New `Node`.

`oldChild` of type `Node`

`Node` to be replaced.

Return Value

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

No Exceptions

`isNodeValid`

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

Parameters

`deep` of type `boolean`

Setting the `deep` flag on causes the `isNodeValid` method to check for the whole subtree of the current node for validity. Setting it to `false` only checks the current node and its immediate child nodes. The `validate` method on the `DocumentAS` [p.34] interface, however, checks to determine whether the entire document is valid.

`wFValidityCheckLevel` of type `unsigned short`

Flag to tell at what level validity and well-formedness checking is done.

Return Value

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

Exceptions

`DOMASException` [p.14]	`NO_AS_AVAILABLE`: Raised if the `DocumentEditAS` [p.37] related to this node does not have any active `ASModel` [p.15] and `wfValidityCheckLevel` is set to `PARTIAL` or `STRICT_VALIDITY_CHECK`.

Interface *ElementEditAS*

This interface extends the `Element` interface with additional methods for guided document editing. An object implementing this interface must also implement NodeEditAS interface.

IDL Definition

```
interface ElementEditAS : NodeEditAS {
  readonly attribute NodeList         definedElementTypes;
  unsigned short     contentType();
  boolean            canSetAttribute(in DOMString attrname,
                                     in DOMString attrval);
  boolean            canSetAttributeNode(in Attr attrNode);
  boolean            canSetAttributeNS(in DOMString name,
                                       in DOMString attrval,
                                       in DOMString namespaceURI);
  boolean            canRemoveAttribute(in DOMString attrname);
  boolean            canRemoveAttributeNS(in DOMString attrname,
                                          in DOMString namespaceURI);
  boolean            canRemoveAttributeNode(in Node attrNode);
  NodeList           getChildElements();
  NodeList           getParentElements();
  NodeList           getAttributeList();
  boolean            isElementDefined(in DOMString elemTypeName);
  boolean            isElementDefinedNS(in DOMString elemTypeName,
                                        in DOMString namespaceURI,
                                        in DOMString name);
};
```

Attributes

`definedElementTypes` of type `NodeList`, readonly

The list of qualified element names defined in the abstract schema.

Methods

`canRemoveAttribute`

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

Parameters

`attrname` of type `DOMString`

Name of attribute.

Return Value

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

No Exceptions

`canRemoveAttributeNS`

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

Parameters

`attrname` of type `DOMString`

Local name of the attribute to be removed.

`namespaceURI` of type `DOMString`

The namespace URI of the attribute to remove.

Return Value

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

No Exceptions

`canRemoveAttributeNode`

Determines if an attribute node can be removed.

Parameters

`attrNode` of type `Node`

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

Return Value

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

No Exceptions

`canSetAttribute`

Determines if the value for specified attribute can be set.

Parameters

`attrname` of type `DOMString`

Name of attribute.

`attrval` of type `DOMString`

Value to be assigned to the attribute.

Return Value

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

No Exceptions

`canSetAttributeNS`

Determines if the attribute with given namespace and qualified name can be created if not already present in the attribute list of the element. If the attribute with same qualified name and namespaceURI is already present in the elements attribute list it tests for the value of the attribute and its prefix to the new value. See DOM core `setAttributeNS`.

Parameters

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

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

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

Return Value

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

No Exceptions

`canSetAttributeNode`

Determines if an attribute node can be added with respect to the validity check level.

Issue canSetAttributeNode-1:
: This is an attribute node, there is no need for canSetAttributreNodeNS!

Parameters

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

Return Value

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

No Exceptions

`contentType`

Determines element content type.

Return Value

`unsigned short`	Constant for one of EMPTY_CONTENTTYPE, ANY_CONTENTTYPE, MIXED_CONTENTTYPE, ELEMENTS_CONTENTTYPE.

No Parameters

No Exceptions

`getAttributeList`

Returns an `NodeList` containing all the possible `Attr`s that can appear with this type of element.

Return Value

`NodeList`	List of possible attributes of this element.

No Parameters

No Exceptions

`getChildElements`

Returns an `NodeList` containing the possible `Element` names that can appear as children of this type of element.

Return Value

`NodeList`	List of possible children element types of this element.

No Parameters

No Exceptions

`getParentElements`

Returns an `NodeList` containing the possible `Element` names that can appear as a parent of this type of element.

Return Value

`NodeList`	List of possible parent element types of this element.

No Parameters

No Exceptions

`isElementDefined`

Determines if this element is defined in the currently active AS.

Parameters

`elemTypeName` of type `DOMString`

Name of element.

Return Value

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

No Exceptions

`isElementDefinedNS`

Determines if this element in this namespace is defined in the currently active AS.

Parameters

`elemTypeName` of type `DOMString`

Name of element.

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

`name` of type `DOMString`
: Qualified name of namespace. This is for sub-elements.

Return Value

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

No Exceptions

Interface *CharacterDataEditAS*

This interface extends the `NodeEditAS` [p.38] interface with additional methods for document editing. An object implementing this interface must also implement NodeEditAS interface.

IDL Definition

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

Attributes

`isWhitespaceOnly` of type `boolean`, readonly
: `true` if content only whitespace; `false` for non-whitespace.

Methods

`canAppendData`
: Determines if data can be appended.

Parameters

`arg` of type `DOMString`
: Argument to be appended.

Return Value

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

No Exceptions

`canDeleteData`
: Determines if data can be deleted.

Parameters

`offset` of type `unsigned long`
Offset.
`count` of type `unsigned long`
Number of 16-bit units to delete.

Return Value

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

No Exceptions

`canInsertData`
Determines if data can be inserted.

Parameters

`offset` of type `unsigned long`
Offset.
`arg` of type `DOMString`
Argument to be set.

Return Value

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

No Exceptions

`canReplaceData`
Determines if data can be replaced.

Parameters

`offset` of type `unsigned long`
Offset.
`count` of type `unsigned long`
Replacement.
`arg` of type `DOMString`
Argument to be set.

Return Value

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

No Exceptions

`canSetData`
Determines if data can be set.

Parameters

`offset` of type `unsigned long`
Offset.
`count` of type `unsigned long`
Argument to be set.

Return Value

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

No Exceptions

1.5. Editing and Generating an Abstract Schema

Editing and generating an abstract schema falls in the AS-editing world. The most obvious requirement for this set of requirements is for tools that author abstract schemas, either under user control, i.e., explicitly designed document types, or generated from other representations. The latter class includes transcoding tools, e.g., synthesizing an XML representation to match a database schema.

It's important to note here that a DTD's "internal subset" is part of the Abstract Schema, yet is loaded, stored, and maintained as part of the individual document instance. This implies that even tools which do not want to let users change the definition of the Document Type may need to support editing operations upon this portion of the AS. It also means that our representation of the AS must be aware of where each portion of its content resides, so that when the serializer processes this document it can write out just the internal subset. A similar issue may arise with external parsed entities, or if schemas introduce the ability to reference other schemas. Finally, the internal-subset case suggests that we may want at least a two-level representation of abstract schemas, so a single DOM representation of a DTD can be shared among several documents, each potentially also having its own internal subset; it's possible that entity layering may be represented the same way.

The *API* [p.135] for altering the abstract schema may also be the AS's official interface with parsers. One of the ongoing problems in the DOM is that there is some information which must currently be created via completely undocumented mechanisms, which limits the ability to mix and match DOMs and parsers. Given that specialized DOMs are going to become more common (sub-classed, or wrappers around other kinds of storage, or optimized for specific tasks), we must avoid that situation and provide a "builder" API. Particular pairs of DOMs and parsers may bypass it, but it's required as a portability mechanism.

Note that several of these applications require that an AS be able to be created, loaded, and manipulated without/before being bound to a specific Document. A related issue is that we'd want to be able to share a single representation of an AS among several documents, both for storage efficiency and so that changes in the AS can quickly be tested by validating it against a set of known-good documents. Similarly, there is a known problem in [DOM Level 3 Core] where we assume that the `DocumentType` will be created before the `Document`, which is fine for newly-constructed documents but not a good match for the order in which an XML parser encounters this data; being able to "rebind" a `Document` to a new AS, after it has been created may be desirable.

As noted earlier, questions about whether one can alter the content of the AS via its syntax, via higher-level abstractions, or both, exist. It's also worth noting that many of the editing concepts from the Document tree still apply; users should probably be able to clone part of an AS, remove and re-insert parts, and so on.

1.6. Abstract Schema-directed Document Manipulation

In addition to using the abstract schema to validate a document instance, applications would like to be able to use it to guide construction and editing of documents, which falls into the document-editing world. Examples of this sort of guided editing already exist, and are becoming more common. The necessary queries can be phrased in several ways, the most useful of which may be a combination of "what does the DTD allow me to insert here" and "if I insert this here, will the document still be valid". The former is better suited to presentation to humans via a user interface, and when taken together with sub-tree validation may subsume the latter.

It has been proposed that in addition to asking questions about specific parts of the abstract schema, there should be a reasonable way to obtain a list of all the defined symbols of a given type (element, attribute, entity) independent of whether they're valid in a given location; that might be useful in building a list in a user-interface, which could then be updated to reflect which of these are relevant for the program's current state.

Remember that namespaces also weigh in on this issue, in the case of attributes, a "can-this-go-there" may prompt a namespace-well-formedness check and warn you if you're about to conflict with or overwrite another attribute with the same namespaceURI/localName but different prefix, or same nodeName but different namespaceURI.

We have to deal with the fact that "the shortest distance between two valid documents may be through an invalid one". Users may want to know several levels of detail (all the possible children, those which would be valid given what precedes this point, those which would be valid given both preceding and following siblings). Also, once XML Schemas introduce context sensitive validity, we may have to consider the effect of children as well as the individual node being inserted.

1.7. Validating a Document Against an Abstract Schema

The most obvious use for an abstract schema (DTD or XML Schema or any Abstract Schema) is to use it to validate that a given XML document is in fact a properly constructed instance of the document type described by this AS. This again falls into the document-editing world. The XML spec only discusses performing this test at the time the document is loaded into the "processor", which most of us have taken to mean that this check should be performed at parse time. But it is obviously desirable to be able to validate again a document -- or selected subtrees -- at other times. One such case would be validating an edited or newly constructed document before serializing it or otherwise passing it to other users. This issue also arises if the "internal subset" is altered -- or if the whole Abstract Schema changes.

In the past, the DOM has allowed users to create invalid documents, and assumed the serializer would accept the task of detecting problems and announcing/repairing them when the document was written out in XML syntax... or that they would be checked for validity when read back in. We considered adding validity checks to the DOM's existing editing operations to prevent creation of invalid documents, but are currently inclined against this for several reasons. First, it would impose a significant amount of computational overhead to the DOM, which might be unnecessary in many situations, e.g., if the change is occurring in a context where we know the result will be valid. Second, "the shortest distance between two good documents may be through a bad document". Preventing a document from becoming temporarily invalid may impose a considerable amount of additional work on higher-level code and users Hence our current plan is to continue to permit editing to produce invalid DOMs, but provide operations which permit a user to check the validity of a node on demand. If needed one can use `continuousValidityChecking` flag to ensure that the DOM remains valid during the editing process.

Note that validation includes checking that ID attributes are unique, and that IDREFs point to IDs which actually exist.

1.8. Well-formedness Testing

XML defined the "well-formed" (*WF*) state for documents which are parsed without reference to their DTDs. Knowing that a document is well-formed may be useful by itself even when a DTD is available. For example, users may wish to deliberately save an invalid document, perhaps as a checkpoint before further editing. Hence, the AS feature will permit both full validity checking (see previous section) and "lightweight" WF checking, as requested by the caller, as well as processing entity declarations in the AS even if validation is not turned on. This falls within the document-editing world.

While the DOM inherently enforces some of XML’s well-formedness conditions (proper nesting of elements, constraints on which children may be placed within each node), there are some checks that are not yet performed. These include:

- Character restrictions for text content and attribute values. Some characters aren’t permitted even when expressed as numeric character entities
- The three-character sequence "]]>" in CDATASections.
- The two-character sequence "--" in comments. (Which, be it noted, some XML validators don’t currently remember to test...)

In addition, Namespaces introduce their own concepts of well-formedness. Specifically:

- No two attributes on a single Element may have the same combination of namespaceURI and localName, even if their prefixes are different and hence they don’t conflict under XML 1.0 rules.
- NamespaceURIs must be legal URI syntax. (Note that once we have this code, it may be reusable for the URI "datatype" in document content; see discussion of datatypes.)
- The mapping of namespace prefixes to their URIs must be declared and consistent. That isn’t required during normal DOM operation, since we perform "early binding" and thereafter refer to nodes primarily via their namespaceURIs and localName. But it does become an issue when we want to serialize the DOM to XML syntax, and may be an issue if an application is assuming that all the declarations are present and correct. This may imply that we should provide a `namespaceNormalize` operation, which would create the implied declarations and reconcile conflicts in some reasonably standardized manner. This may be a major undertaking, since some DOMs may be using the namespace to direct subclassing of the nodes or similar special treatment; as with the existing `normalize` method, you may be left with a different-but-equivalent set of node objects.

In the past, the DOM has allowed users to create documents which violate these rules, and assumed the serializer would accept the task of detecting problems and announcing/repairing them when the document was written out in XML syntax. We considered adding WF checks to the DOM's existing editing operations to prevent WF violations from arising, but are currently inclined against this for two reasons. First, it would impose a significant amount of computational overhead to the DOM, which might be unnecessary in many situations (for example, if the change is occurring in a context where we know the illegal characters have already been prevented from arising). Second, "the shortest distance between two good documents may be through a bad document" -- preventing a document from becoming temporarily ill-formed may impose a considerable amount of additional work on higher-level code and users. (Note possible issue for Serialization: In some applications, being able to save and reload marginally poorly-formed DOMs might be useful -- editor checkpoint files, for example.) Hence our current plan is to continue to permit editing to produce ill-formed DOMs, but provide operations which permit a user to check the well-formedness of a node on demand, and possibly provide some of the primitive (e.g., string-checking) functions directly.

1.9. Load and Save for Abstract Schemas

The module extends the Document Object Model Load and Save [p.53] module to permit to load a `Document` using a specific `ASModel` [p.15] and to load an `ASModel` from an URI or `DOMInputSource` [p.81] .

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "AS-LS" and "3.0" (respectively) to determine whether or not the Load and Save for Abstract Schemas module is supported by the implementation. In order to fully support this module, an implementation must also support the "`AS-EDIT`" features defined in this specification.

Interface *DOMASBuilder*

An Abstract Schema parser interface.

`DOMASBuilder` provides an API for parsing Abstract Schemas and building the corresponding `ASModel` [p.15] tree.

IDL Definition

```
interface DOMASBuilder : ls::DOMBuilder {
           attribute ASModel          abstractSchema;
  ASModel            parseASURI(in DOMString uri)
                                        raises(DOMASException, 
                                               DOMSystemException);
  ASModel            parseASInputSource(in ls::DOMInputSource is)
                                        raises(DOMASException, 
                                               DOMSystemException);
};
```

Attributes

`abstractSchema` of type `ASModel` [p.15]

Associate an `ASModel` [p.15] with a `DOMBuilder` [p.66] . This `ASModel` will be used by the "`validate-if-schema`" and "`datatype-normalization`" options during the load of a new `Document`.

Methods

`parseASInputSource`

Parse a Abstract Schema from a location identified by an `DOMInputSource` [p.81] .

Parameters

`is` of type `ls::DOMInputSource`

The `DOMInputSource` [p.81] from which the source Abstract Schema is to be read.

Return Value

`ASModel` [p.15]	The newly created `ASModel`.

Exceptions

`DOMASException` [p.14]	Exceptions raised by `parseASURI()` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default error handlers will raise a `DOMASException` [p.14] if any form of Abstract Schema inconsistencies or warning occurs during the parse, but application defined errorHandlers are not required to do so. Raise a WRONG_MIME_TYPE_ERR when `mimeTypeCheck` is `true` and the inputsource has an incorrect MIME Type. See attribute `mimeTypeCheck`.
`DOMSystemException`	Exceptions raised by `parseURI()` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default error handlers will raise a DOMSystemException if any form I/O or other system error occurs during the parse, but application defined error handlers are not required to do so.

`parseASURI`

Parse a Abstract Schema from a location identified by an URI reference.

Parameters

`uri` of type `DOMString`

The location of the Abstract Schema to be read.

Return Value

`ASModel` [p.15]	The newly created Abstract Schema.

Exceptions

`DOMASException` [p.14]	Exceptions raised by `parseASURI()` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default error handlers will raise a `DOMASException` [p.14] if any form of Abstract Schema inconsistencies or warning occurs during the parse, but application defined errorHandlers are not required to do so. WRONG_MIME_TYPE_ERR: Raised when `mimeTypeCheck` is `true` and the input source has an incorrect MIME Type. See the attribute `mimeTypeCheck`.
`DOMSystemException`	Exceptions raised by `parseURI()` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default error handlers will raise a DOMSystemException if any form I/O or other system error occurs during the parse, but application defined error handlers are not required to do so.

Interface *DOMASWriter*

A Abstract Schema serialization interface.

DOMASWriters provides an API for serializing Abstract Schemas out in the form of a source Abstract Schema. The Abstract Schema is written to an output stream, the type of which depends on the specific language bindings in use.

DOMASWriter is a generic Abstract Schema serialization interface. It can be applied to both an internal Abstract Schema and/or an external Abstract Schema. DOMASWriter is applied to serialize a single Abstract Schema. Serializing a document with an active Internal Abstract Schema will serialize this internal Abstract Schema with the document as it is part of the Document (see `DOMWriter` [p.74]).

IDL Definition

```
interface DOMASWriter : ls::DOMWriter {
  void               writeASModel(in DOMOutputStream destination,
                                  in ASModel model)
                                        raises(DOMSystemException);
};
```

Methods

`writeASModel`

Write out the specified Abstract Schema to the specified destination.

Issue writeASModel-1:

Does it write a DTD or an XML Schema (or something else)? Is it possible to use this method to convert a DTD to an XML Schema?

Parameters

`destination` of type `DOMOutputStream`

The destination for the data to be written.

`model` of type `ASModel` [p.15]

The Abstract Schema to serialize.

Exceptions

`DOMSystemException`	This exception will be raised in response to any sort of IO or system error that occurs while writing to the destination. It may wrap an underlying system exception.

No Return Value

2. Document Object Model Load and Save

Editors:
Jeroen van Rotterdam, X-Hive Corporation
Johnny Stenback, Netscape
Andy Heninger, IBM (until March 2001)

2.1. Load and Save Requirements

DOM Level 3 will provide an *API* [p.135] for loading XML documents into a DOM representation and for saving a DOM representation as a XML document.

Some environments, such as the Java [Java] or COM [COM], have their own ways to persist objects to streams and to restore them. There is no direct relationship between these mechanisms and the DOM load/save mechanism. This specification defines how to serialize documents only to and from XML format.

2.1.1. General Requirements

Requirements that apply to both loading and saving documents.

2.1.1.1. Document Sources

Documents must be able to be parsed from and saved to the following sources:

- Input and Output Streams
- URIs
- Files

Note that Input and Output streams take care of the in memory case. One point of caution is that a stream doesn’t allow a base URI to be defined against which all relative URIs in the document are resolved.

2.1.1.2. Abstract Schema Loading

While creating a new document using the DOM API, a mechanism must be provided to specify that the new document uses a pre-existing Abstract Schema and to cause that Abstract Schema to be loaded.

Note that while DOM Level 2 creation can specify a Abstract Schema when creating a document (public and system IDs for the external subset, and a string for the subset), DOM Level 2 implementations do not process the Abstract Schema’s content. For DOM Level 3, the Abstract Schema’s content must be read.

2.1.1.3. Abstract Schema Reuse

When processing a series of documents, all of which use the same Abstract Schema, implementations should be able to reuse the already parsed and loaded Abstract Schema rather than parsing it again for each new document.

This feature may not have an explicit DOM API associated with it, but it does require that nothing in this section, or the Abstract Schema section, of this specification block it or make it difficult to implement.

2.1.1.4. Entity Resolution

Some means is required to allow applications to map public and system IDs to the correct document. This facility should provide sufficient capability to allow the implementation of catalogs, but providing catalogs themselves is not a requirement. In addition XML Base needs to be addressed.

2.1.1.5. Error Reporting

Loading a document can cause the generation of errors including:

- I/O Errors, such as the inability to find or open the specified document.
  XML well formedness errors.
  Validity errors

Saving a document can cause the generation of errors including:

- I/O Errors, such as the inability to write to a specified stream, URI, or file.
  Improper constructs, such as '--' in comments, in the DOM that cannot be represented as well formed XML.

This section, as well as the DOM Level 3 Abstract Schema section should use a common error reporting mechanism. Well-formedness and validity checking are in the domain of the Abstract Schema section, even though they may be commonly generated in response to an application asking that a document be loaded.

2.1.2. Load Requirements

The following requirements apply to loading documents.

2.1.2.1. Parser Properties and Options

Parsers may have properties or options that can be set by applications. Examples include:

- Expansion of entity references.
- Creation of entity ref nodes.
- Handling of white space in element content.
- Enabling of namespace handling.
- Enabling of abstract schema validation.

A mechanism to set properties, query the state of properties, and to query the set of properties supported by a particular DOM implementation is required.

2.1.3. XML Writer Requirements

The fundamental requirement is to write a DOM document as XML source. All information to be serialized should be available via the normal DOM API.

2.1.3.1. XML Writer Properties and Options

There are several options that can be defined when saving an XML document. Some of these are:

- Saving to Canonical XML format.
- Pretty Printing.
- Specify the encoding in which a document is written.
- How and when to use character entities.
- Namespace prefix handling.
- Saving of Abstract Schemas.
- Handling of external entities.

2.1.3.2. Abstract Schema Saving

Requirement from the Abstract Schema group.

2.1.4. Other Items Under Consideration

The following items are not committed to, but are under consideration. Public feedback on these items is especially requested.

2.1.4.1. Incremental and/or Concurrent Parsing

Note: This is done with the asynch loading.

Provide the ability for a thread that requested the loading of a document to continue execution without blocking while the document is being loaded. This would require some sort of notification or completion event when the loading process was done.

Provide the ability to examine the partial DOM representation before it has been fully loaded.

In one form, a document may be loaded asynchronously while a DOM based application is accessing the document. In another form, the application may explicitly ask for the next incremental portion of a document to be loaded.

2.1.4.2. Filtered Save

Provide the capability to write out only a part of a document. May be able to leverage TreeWalkers, or the Filters associated with TreeWalkers, or Ranges as a means of specifying the portion of the document to be written.

2.1.4.3. Document Fragments

Note: Won’t happen.

Document fragments, as specified by the XML Fragment specification, should be able to be loaded. This is useful to applications that only need to process some part of a large document. Because the DOM is typically implemented as an in-memory representation of a document, fully loading large documents can require large amounts of memory.

XPath should also be considered as a way to identify XML Document fragments to load.

2.1.4.4. Document Fragments in Context of Existing DOM

Document fragments, as specified by the XML Fragment specification, should be able to be loaded into the context of an existing document at a point specified by a node position, or perhaps a range. This is a separate feature than simply loading document fragments as a new Node.

2.2. Issue List

2.2.1. Open Issues

Issue LS-Issue-53:
: "format-canonical" needs a correct reference to the spec for canonical XML.

Issue LS-Issue-54:
: How should default attributes be dealt with wrt DOMBuilderFilter?

Issue LS-Issue-55:
: Should we make it possible to `SKIP` an element in `DOMBuilderFilter::endNode`?

Issue LS-Issue-56:
: `namespaceURI` in core can be empty string, how should that be dealt with in DOM LS?

Issue LS-Issue-155:
: This is not really an issue, it’s left in here as a reminder for now. The ’feature’ vs. ’option’ vs. ’property’ mess needs to be cleaned up.

2.2.2. Resolved Issues

Issue LS-Issue-1:
: Should these methods be in a new interface, or should they be added to the existing DOMImplementation Interface? I think that adding them to the existing interface is cleaner, because it helps avoid an explosion of new interfaces.
: The methods are in a separate interface in this description for convenience in preparing the doc, so that I don’t need to edit Core to add the methods. (The same argument could perhaps be made for implementations.)
: **Resolution:** The methods are in a separate DOMImplementationLS interface. Because Load/Save is an optional module, we don’t want to add its to the core DOMImplementation interface.

Issue LS-Issue-2:

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

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

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

This issue was revisited once again at the 9/2000 meeting. Now all DOM properties or features will be short, descriptive names, and we will recommend that all vendor-specific extensions be prefixed to avoid collisions, but will not make specific recommendations for the syntax of the prefix.

Issue LS-Issue-3:

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

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

Issue LS-Issue-4:

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

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

Issue LS-Issue-5:

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

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

Issue LS-Issue-6:

The relationship between SAXExceptions and DOM exceptions seems confusing.

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

Issue LS-Issue-7:

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

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

Issue LS-Issue-8:

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

This question was discussed at the July f2f, without resolution. Agreed that the feature is not critical to the rest of the API, and can be postponed.
Resolution: The issue is moving to core, where it is part of the bigger question of where does the DOM implementation come from, and how do multiple implementations coexist. Allowing separate, or mix-and-match, specification of the parser and the rest of the DOM is not generally practical because parsers generally have some degree of private knowledge about their DOMs.

Issue LS-Issue-9:

The use of interfaces from SAX2 raises some questions. The Java bindings for these interfaces need to be exactly the SAX2 definitions, including the original org.xml.sax package name.
The IDL presented here for these interfaces is an attempt to map the Java into IDL, but it will certainly not round-trip accurately - Java bindings generated from the IDL will not match the original Java.
The reasons for using the SAX interfaces are that they are well designed, widely implemented and used, and provide what is needed. Designing something new would create confusion for application developers (which should be used?) and make extra work for implementers of the DOM, most of whom probably already provide SAX, all for no real gain.
Resolution: Problem is gone. We are not using SAX2. The design will borrow features and concepts from SAX2 when it makes sense to do so.

Issue LS-Issue-10:

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

Issue LS-Issue-11:

Another Error Reporting Question. We decided at the June f2f that validity errors should not be exceptions. This means that a document load operation could encounter multiple errors. Should these be collected and delivered as some sort of collection at the (otherwise) successful completion of the load, or should there be some sort of callback? Callbacks are harder for applications to deal with.
Resolution: Provide a callback mechanism. Provide a default error handler that throws an exception and stops further processing. From July f2f.

Issue LS-Issue-12:

Definition of "Non-validating". Exactly how much processing is done by "non-validating" parsers is not fully defined by the XML specification. In particular, they are not required to read any external entities, but are not prohibited from doing so.
Another common user request: a mode that completely ignores DTDs, both and external. Such a parser would not conform to XML 1.0, however.
For the documents produced by a non-validating load to be the same, we need to tie down exactly what processing must be done. The XML Core WG also has question as an open issue .
Some discussion is at http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000JanMar/0192.html
Here is proposal: Have three classes of parsers

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

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

Issue LS-Issue-13:
: Use of System or Language specific types for Input and Output
Loading and Saving requires that one of the possible sources or destinations of the XML data be some sort of stream that can be used with io streams or memory buffers, or anything else that might take or supply data. The type will vary, depending on the language binding.
The question is, what should be put into the IDL interfaces for these? Should we define an XML stream to abstract out the dependency, or use system classes directly in the bindings?
Resolution: Define IDL types for use in the rest of the interface definitions. These types will be mapped directly to system types for each language binding

Issue LS-Issue-14:
: Should there be separate DOM modules for browser or scripting style loading (document.load("whatever")) and server style parsers? It's probably easy for the server style parsers to implement the browser style interface, but the reverse may not be `true`.
Resolution: Yes. A client application style API will be provided.

Issue LS-Issue-15:
: System Exceptions. Loading involves file opens and reads, and these can result in a variety of system errors that may already have associated system exceptions. Should these system exceptions pass through as is, or should they be some how wrapped in DOMExceptions, or should there be a parallel set DOM Exceptions, or what?
Resolution: Introduce a new DOMSystemException to standardize the reporting of common I/O errors across different DOM environments. Let it wrap an underlying system exception or error code when appropriate. To be defined in the common ErrorReporting module, to be shared with Abstract Schema.

Issue LS-Issue-16:
: Loading and saving of abstract schema's - DTDs or Schemas - outside of the context of a document is not addressed.
Resolution: See the `DOMASBuilder` [p.49] interface in the AS spec

Issue LS-Issue-17:
: Loading while validating using an already loaded abstract schema is not addressed. Applications should be able to load a abstract schema (issue 16), and then repeatedly reuse it during the loading of additional documents.
Resolution: See the `DOMASBuilder` [p.49] interface in the AS spec

Issue LS-Issue-18:
: For the list of parser properties, which must all implementations recognize, which settings must all implementations support, and which are optional?
Resolution: Done

Issue LS-Issue-19:
: DOMOutputStream: should this be an interface with methods, or just an opaque type that maps onto an appropriate binding-specific stream type?
If we specify an actual interface with methods, applications can implement it to wrap any arbitrary destination that they may have. If we go with the system type it's simpler to output to that type of stream, but harder otherwise.
Resolution: Opaque.

Issue LS-Issue-20:
: Action from September f2f to "add issues raised by schema discussion. What were these?
Resolution: nobody seems to remember this, no action taken

Issue LS-Issue-21:

Define exceptions. A `DOMSystemException` needs to be defined as part of the error handling module that is to be shared with AS. Common I/O type errors need to be defined for it, so that they can be reported in a uniform way. A way to embed errors or exceptions from the OS or language environment is needed, to provide full information to applications that want it.
Resolution: Duplicate of issue #15

Issue LS-Issue-22:

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

Issue LS-Issue-23:

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

Issue LS-Issue-24:

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

Issue LS-Issue-25:

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

Issue LS-Issue-26:

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

Issue LS-Issue-27:

How is validation during document loading handled when there are multiple possible abstract schemas associated with the document? How is one selected? The same question exists for documents in general, outside of the context of loading. Resolving the question for loading probably needs to wait until the more general question is understood.
Resolution: Always use the active external AS if any and the active internal AS if any. Whenever you want to validate during parsing with a different Internal/External model you have to activate this Abstract Schema first.

Issue LS-Issue-29:

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

Issue LS-Issue-28:

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

createEntityReferenceNodes to be `true`.
(***ED:*** Is this really what we want?)
Resolution: new feature added.

Issue LS-Issue-30:
Possible additional parser features - option to not create CDATA nodes, and to merge CDATA contents with adjacent TEXT nodes if they exist. Otherwise just create a TEXT node.
Option to omit Comments.
Resolution: new feature added.

Issue LS-Issue-31:
We now have an option for fixing up namespace declarations and prefixes on serialization. Should we specify how this is done, so that the documents from different implementations of serialization will use the same declarations and prefixes, or should we leave the details up to the implementation?
Resolution: The exact form of the namespace fixup is implementation dependent. The only requirement is that all elements and attributes end up with the correct namespace URI.

Issue LS-Issue-32:
Mimetypes. If the input being parsed is from http or something else that supplies types, and the type is something other than text/xml, should we parse it anyhow, or should we complain. Should there be an option?
Tentative resolution: always parse, never complain. Reasons: 1. This is what all parsers do now, and no one has ever complained, at least not that I'm aware of. 2. Applications must have a pretty good reason to suspect that they're getting xml or they wouldn't have invoked the parser. 3. All the test would do is to take something that might have worked (xml that is not known to the server) and turn it into an error. Non-xml is exceptionally unlikely to successfully parse (be well formed.)
Resolution: See the mimeTypeCheck attribute on `DOMBuilder` [p.66] .

Issue LS-Issue-33:
Unicode Character Normalization Problems. It turns out that for some code pages, normalizing a Unicode representation, translating to the code page, then translating back to Unicode can result in un-normalized Unicode. Mark Davis says that this can happen with Vietnamese and maybe with Hebrew.
This means that the suggested W3C model of normalization on serialization (early normalization) may not work, and that the receiver of the data may need to normalize it again, just in case.
Resolution: The scenario described is a quality-of-implementation issue. A transcoder converting from the one of the troublesome code pages to a Unicode representation should be responsible for re-normalizing the output.

Issue LS-Issue-34:
Features 2.1.4.1, 2 - XML Fragment Support. Should these be dropped?
Resolution: The DOM WG decided to drop support for XML fragment loading in the DOM Level 3 Load-Save module due to lack of time to define the behavior in all the edge cases, future versions of this spec might address this issue.

Issue LS-Issue-35:
XPath based document load filter. It would be plausible to have a partial (filtered) document load based on selecting the portion of the document to load with an XPath expression. This facility could be in addition to the node-by-node filtering currently specified. Or we could drop the existing filter. Implementing an XPath based selective load would require that there be an XPath processor present in addition to the parser itself.
Resolution: The DOM Level 3 spec will not define an interface for doing XPath/XPointer type

filtering, implementations are free to implement XPath/XPointer based filters on top of a DOMBuilderFilter.

Issue LS-Issue-36:

MIME Type checking for DOMASBuilder.

What MIME Type checking needs to be done for parsing schemas

Resolution: see DOMBuilder, DOMASBuilder is an extend of DOMBuilder, this issue is solved within DOMBuilder

Issue LS-Issue-37:

Internal ASModel serialization for DOMWriter.

What if the internal ASModel is an XML Schema ASModel. Currently there is no ASModel type. Adding an Internal ASModel can be any kind of schema. Should serialization somehow check the internal ASModel ? What about the internal subset, is it discarded when the AS spec is implemented ?

Resolution: An internal ASModel can’t be a schema according to the AS spec. The internal subset is discarded when an Abstract Schema is active and the AS spec is implemented

Issue LS-Issue-38:

Attribute Normalization.

Add a property to "attributeNormalization" to DOMWriter to support or discard Attribute Normalization during serialization to. Setting attributeNormalization will serialize attributes with unexpanded entity references (if any) regardless their childnode(s). This means that if a user is changing the child nodes of an entity reference node within an attribute and attributeNormalization is set to `true` during serialization that these changes are discarded during serialization.

Resolution: The normalization will be driven by the validation options on DOMBuilder, if a document is validated it will also be normalized, if the document is not validated then no normalization will occure.

Issue LS-Issue-39:

Validation at serialization time. Should we have an option for validating while serializing, what about validation errors, should we allow serializing non-valid DOM’s?

Resolution: No. Validation at serialization time will not be supported by this specification.

Issue LS-Issue-40:

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

Resolution: Yes, the description is acceptable.

Issue LS-Issue-41:

Do we need filter support in DOMWriter too?

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

Issue LS-Issue-42:

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

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

Issue LS-Issue-43:

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

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

Issue LS-Issue-44:

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

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

Issue LS-Issue-45:

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

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

Issue LS-Issue-46:

Should supporting async/sync loading be optional?

Resolution: Yes.

Issue LS-Issue-47:

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

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

Issue LS-Issue-48:

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

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

Issue LS-Issue-49:

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

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

Issue LS-Issue-50:

Should ’external-parameter-entities’ be replaced by an "load-external-dtds-n’-stuff" option?

Resolution: yes, done, "external-parameter-entities" added.

Issue LS-Issue-51:

DOMBuilder::canSetFeature and ::supportsFeature are redundant, no?

Resolution: Yes, supportsFeature removed.

Issue LS-Issue-52:

Is the API dependencies on the Events spec acceptable?

Resolution: We’re only reusing events API’s, we’re not requiring people to implement the events spec so this shouldn’t be a problem.

Issue LS-ISSUE-53:

Doesn’t the feature "external-dtd-subset" conflict with the XML 1.0 specifications standalone="true"?

Resolution: No, the standalone "attribute" in XML 1.0 is only a hint, and thus implementations are not required to do anything with it that matters for a DOM builder.

2.3. Interfaces

This section defines an *API* [p.135] for loading (parsing) XML documents [XML] into a DOM representation [DOM Level 3 Core] and for saving (serializing) a DOM representation as an XML document.

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

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

- `DOMImplementationLS` [p.64] -- A new `DOMImplementation` interface that provides the factory methods for creating the objects required for loading and saving.
- `DOMBuilder` [p.66] -- A parser interface.
- `DOMInputSource` [p.81] -- Encapsulate information about the XML document to be loaded.
- `DOMEntityResolver` [p.83] -- During loading, provides a way for applications to redirect references to external entities.
- `DOMBuilderFilter` [p.85] -- Provide the ability to examine and optionally remove Element nodes as they are being processed during the parsing of a document.
- `DOMWriter` [p.74] -- An interface for writing out or serializing DOM documents.
- `DocumentLS` [p.87] -- Provides a client or browser style interface for loading and saving.
- `ParseErrorEvent` [p.89] -- ParseErrorEvent is the event that is fired if there’s an error in the XML document being parsed using the methods of DocumentLS.

Interface *DOMImplementationLS*

`DOMImplementationLS` contains the factory methods for creating objects that implement the `DOMBuilder` [p.66] (parser) and `DOMWriter` [p.74] (serializer) interfaces.

An object that implements DOMImplementationLS is obtained by doing a binding specific cast from DOMImplementation to DOMImplementationLS. Implementations supporting the Load and Save feature must implement the DOMImplementationLS interface on whatever object implements the DOMImplementation interface.

IDL Definition

```
interface DOMImplementationLS {

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

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

Definition group *DOMIMplementationLSMode*

An integer indicating which type of mode this is.

Defined Constants

`MODE_ASYNCHRONOUS`
: Create an asynchronous `DOMBuilder` [p.66] .

`MODE_SYNCHRONOUS`
: Create a synchronous `DOMBuilder` [p.66] .

Methods

`createDOMBuilder`

Create a new `DOMBuilder` [p.66] . The newly constructed parser may then be configured by means of its `setFeature` method, and used to parse documents by means of its `parse` method.

Parameters

`mode` of type `unsigned short`
: The `mode` argument is either `MODE_SYNCHRONOUS` or `MODE_ASYNCHRONOUS`, if `mode` is `MODE_SYNCHRONOUS` then the `DOMBuilder` [p.66] that is created will operate in synchronous mode, if it’s `MODE_ASYNCHRONOUS` then the `DOMBuilder` that is created will operate in asynchronous mode.

Return Value

`DOMBuilder` [p.66]	The newly created `DOMBuilder` object, this `DOMBuilder` is either synchronous or asynchronous depending on the value of the `type` argument.

Exceptions

`DOMException`	Raise a NOT_SUPPORTED_ERR exception if MODE_ASYNCHRONOUS is not supported.

`createDOMInputSource`

Create a new "empty" `DOMInputSource` [p.81] .

Return Value

`DOMInputSource` [p.81]	The newly created `DOMBuilder` [p.66] object, this `DOMBuilder` is either synchronous or asynchronous depending on the value of the `type` argument.

No Parameters

No Exceptions

`createDOMWriter`

Create a new `DOMWriter` [p.74] object. `DOMWriter`s are used to serialize a DOM tree back into an XML document.

Return Value

`DOMWriter` [p.74]	The newly created `DOMWriter` object.

No Parameters
No Exceptions

Interface *DOMBuilder*

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

`DOMBuilder` provides an API for parsing XML documents and building the corresponding DOM document tree. A `DOMBuilder` instance is obtained from the `DOMImplementationLS` [p.64] interface by invoking its `createDOMBuilder`method.

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

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

The Document Object Model Level 3 Load and Save does not provide a way to disable the namespace resolution: Namespaces are always taken into account during loading and saving operations.

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

Events supported by asynchronous `DOMBuilder` are:

- **ls-load**: The document that’s being loaded is completely parsed, see the definition of `LSLoadEvent` [p.83]
- **ls-progress**: Progress notification, see the definition of `LSProgressEvent` [p.83]

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

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

`"namespace-declarations"`

`true`

[*required*] (*default*)
include the namespace declaration attributes, specified or defaulted from the schema or the DTD, in the DOM document. See also the section *Declaring Namespaces* in [XML Namespaces].

`false`

[*optional*]
discard all namespace declaration attributes. The Namespace prefixes will be retained even if this feature is set to `false`.

`"validation"`

`true`

[*optional*]
report validation errors (setting `true` also will force the `external-general-entities` and `external-parameter-entities` features to be `true`.) Also note that the `validate-if-schema` feature will alter the validation behavior when this feature is set `true`.

`false`

[*required*] (*default*)
do not report validation errors.

`"external-parameter-entities"`

`true`

[*required*] (*default*)
load external parameter entities.

`false`

[*optional*]
do not load external parameter entities.

default value

`true`

`"external-general-entities"`

`true`

[*required*] (*default*)
include all external general (text) entities.

`false`

[*optional*]
do not include external general entities.

`"external-dtd-subset"`

`true`

[*required*] (*default*)
load the external dtd and also all external parameter entities.

`false`

[*optional*]
do not load the dtd nor external parameter entities.

`"validate-if-schema"`

`true`

[*optional*]
when both this feature and validation are `true`, enable validation only if the document being processed has a schema (i.e. XML schema, DTD, any other type of schema, note that this is unrelated to the abstract schema specification). Documents without schemas are parsed without validation.

`false`

[*required*] (*default*)
the validation feature alone controls whether the document is checked for validity. Documents without a schemas are not valid.

`"validate-against-dtd"`

`true`

[*optional*]
Prefere validation against the DTD over any other schema referenced in the XML file.

`false`

[*required*] (*default*)
Let the parser decide what to validate against if there are references to multiple types of schemas.

`"datatype-normalization"`

`true`

[*required*]
Let the (non-DTD) validation process do its datatype normalization that is defined in the used schema language.
Issue normalization-1:
We should define "datatype normalization".

`false`

[*required*] (*default*)
Disable datatype normalization. The XML 1.0 attribute value normalization is garantee to occur in that case.

`"create-entity-ref-nodes"`

`true`

[*required*] (*default*)
Create `EntityReference` nodes in the DOM document. It will also set `create-entity-nodes` to be `true`.

`false`

[*optional*]
omit all `EntityReference` nodes from the DOM document, putting the entity expansions directly in their place. `Text` nodes are into "normal" form.

`EntityReference` nodes to non-defined entities will still be created in the DOM document.

`"create-entity-nodes"`

`true`

[*required*] (*default*)
Create `Entity` nodes in the DOM document.

`false`

[*optional*]
Omit all `entity` nodes from the DOM document. It will also set `create-entity-ref-nodes` to `false`.

`"whitespace-in-element-content"`

`true`

[*required*] (*default*)
Include white space characters appearing within element content (see [XML] 2.10 "White Space Handling").

`false`

[*optional*]
Omit white space characters appearing within element content. Note that white space characters within element content will only be omitted if it can be identified as such, and not all parsers may be able to do so (see [XML] 2.10 "White Space Handling").

`"create-cdata-nodes"`

`true`

[*required*] (*default*)
Create `CDATASection` nodes in response to the appearance of CDATA sections in the XML document.

`false`

[*optional*]
Do not create `CDATASection` nodes in the DOM document.
The content of any CDATA sections in the XML document appears in the DOM as if it had been normal (non-CDATA) content. If a CDATA section is adjacent to other content, the combined content appears in a single `Text` node, i.e. the `Text` nodes are into "normal" form.

`"comments"`

`true`

[*required*] (*default*)
Include XML comments in the DOM document.

`false`

[*required*]
Discard XML comments, do not create `Comment` nodes in the DOM Document resulting from a parse.

`"charset-overrides-xml-encoding"`

`true`

[*required*] (*default*)
If a higher level protocol such as HTTP [RFC2616] provides an indication of the character encoding of the input stream being processed, that will override any encoding specified in the XML declaration or the Text declaration (see also [XML] 4.3.3 "Character Encoding in

Entities"). Explicitly setting an encoding in the `DOMInputSource` [p.81] overrides encodings from the protocol.

`false`

[*required*]

Any character set encoding information from higher level protocols is ignored by the parser.

`"load-as-infoset"`

`true`

[*optional*]

Load the document and store only the information defined in the XML Information Set [XML Information set].

This will force the following features to `false`: `namespace-declarations`, `validate-if-schema`, `create-entity-ref-nodes`, `create-entity-nodes`, `create-cdata-nodes`.

This will force the following features to `true`: `datatype-normalization`, `whitespace-in-element-content`, `comments`, `charset-overrides-xml-encoding`.

Other features are not changed unless explicity specified in the description of the features.

Note that querying this feature with `getFeature` will return `true` only if the individual features specified above are appropriately set.

`false`

Setting `load-as-infoset` to `false` has no effect.

`"supported-mediatypes-only"`

`true`

[*optional*]

Check that the media type of the parsed resource is a supported media type and call the error handler if an unsupported media type is encountered. The media types defined in [RFC3023] must be accepted.

`false`

[*required*] (*default*)

Don’t check the media type, accept any type of data.

IDL Definition

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

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

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

Definition group *ACTION_TYPES*

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

Defined Constants

`ACTION_APPEND`
: Append the result of parsing the input source to the context node. For this action to work, the context node must be an `Element`.

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

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

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

Attributes

`entityResolver` of type `DOMEntityResolver` [p.83]
: If a `DOMEntityResolver` [p.83] has been specified, each time a reference to an external entity is encountered the `DOMBuilder` will pass the public and system IDs to the entity resolver, which can then specify the actual source of the entity.

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

`filter` of type `DOMBuilderFilter` [p.85]
: When the application provides a filter, the parser will call out to the filter at the completion of the construction of each `Element` node. The filter implementation can choose to remove the element from the document being constructed (unless the element is the document element) or to terminate the parse early. If the document is being validated when it's loaded the validation happens before the filter is called.

Methods

`canSetFeature`

Query whether setting a feature to a specific value is supported.
The feature name has the same form as a DOM hasFeature string.

Parameters

`name` of type `DOMString`
The feature name, which is a DOM has-feature style string.

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

Return Value

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

No Exceptions

`getFeature`

Look up the value of a feature.
The feature name has the same form as a DOM hasFeature string

Parameters

`name` of type `DOMString`
The feature name, which is a string with DOM has-feature syntax.

Return Value

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

Exceptions

`DOMException`	Raise a NOT_FOUND_ERR When the `DOMBuilder` does not recognize the feature name.

`parse`

Parse an XML document from a resource identified by an `DOMInputSource` [p.81] .

Parameters

`is` of type `DOMInputSource` [p.81]
The `DOMInputSource` from which the source document is to be read.

Return Value

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

Exceptions

`DOMSystemException`	Exceptions raised by `parse` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default ErrorHandlers will raise a `DOMSystemException` if any form I/O or other system error occurs during the parse, but application defined ErrorHandlers are not required to do so.

`parseURI`

Parse an XML document from a location identified by an URI reference [RFC2396]. If the URI contains a fragment identifier (see section 4.1 in [RFC2396]), the behavior is not defined by this specification.

Parameters

`uri` of type `DOMString`

The location of the XML document to be read.

Return Value

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

Exceptions

`DOMSystemException`	Exceptions raised by `parseURI` originate with the installed ErrorHandler, and thus depend on the implementation of the `DOMErrorHandler` interfaces. The default error handlers will raise a DOMSystemException if any form I/O or other system error occurs during the parse, but application defined error handlers are not required to do so.

`parseWithContext`

Parse an XML document or fragment from a resource identified by an `DOMInputSource` [p.81] and insert the content into an existing document at the position epcified with the `contextNode` and `action` arguments. When parsing the input stream the context node is used for resolving unbound namespace prefixes.

Parameters

`is` of type `DOMInputSource` [p.81]

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

`cnode` of type `Node`

The `Node` that is used as the context for the data that is being parsed.

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

Exceptions

`DOMException`	HIERARCHY_REQUEST_ERR: Thrown if this action results in an invalid hierarchy (i.e. a Document with more than one document element).

No Return Value

`setFeature`

Set the state of a feature.

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

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

Parameters

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

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

Exceptions

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

No Return Value

Interface *DOMWriter*

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

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

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

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

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 &lt; and &amp. The other predefined entities (&gt, &apos, etc.) are not used; these characters can be included directly. Any character that can not be represented directly in the output character encoding is serialized as a numeric character reference.

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

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

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

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

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

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

While serializing a document the serializer will write out non-specified values (such as attributes whose `specified` is `false`) if the `output-default-values` feature is set to `true`. If the `output-default-values` flag is set to `false` and the `use-abstract-schema` feature is set to `true` the abstract schema will be used to determine if a value is specified or not, if `use-abstract-schema` is not set the `specified` flag on attribute nodes is used to determine if attribute values should be written out.

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

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

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

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

`"normalize-characters"`

  `true`

    [*optional*] (*default*)
    Perform the W3C Text Normalization of the characters [CharModel] in document as they are written out. Only the characters being written are (potentially) altered. The DOM document itself is unchanged.

  `false`

    [*required*]
    do not perform character normalization.

`"split-cdata-sections"`

  `true`

    [*required*] (*default*)
    Split CDATA sections containing the CDATA section termination marker ']]>' or characters that can not be represented in the output encoding, and output the characters using numeric character references. If a CDATA section is split a warning is issued.

`false`

[*required*]
Signal an error if a `CDATASection` contains an unrepresentable character.

`"validation"`

`true`

[*optional*]
Use the abstract schema to validate the document as it is being serialized. If validation errors are found the error handler is notified about the error. Setting this state will also set the feature `use-abstract-schema` to `true`.

`false`

[*required*] (*default*)
Don't validate the document as it is being serialized.

`"expand-entity-references"`

`true`

[*optional*]
Expand `EntityReference` nodes when serializing.

`false`

[*required*] (*default*)
Serialize all `EntityReference` nodes as XML entity references.

`"whitespace-in-element-content"`

`true`

[*required*] (*default*)
Output all white spaces in the document.

`false`

[*optional*]
Only output white space that is not within element content. The implementation is expected to use the `isWhitespaceInElementContent` flag on `Text` nodes to determine if a text node should be written out or not.

`"discard-default-content"`

`true`

[*required*] (*default*)
Use whatever information available to the implementation (i.e. XML schema, DTD, the `specified` flag on `Attr` nodes, and so on) to decide what attributes and content should be serialized or not. Note that the `specified` flag on `Attr` nodes in itself is not always reliable, it is only reliable when it is set to `false` since the only case where it can be set to `false` is if the attribute was created by a Level 1 implementation.

`false`

[*required*]
Output all attributes and all content.

`"format-canonical"`

`true`

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

`false`

[*required*] (*default*)

Don’t canonicalize the output.

`"format-pretty-print"`

`true`

[*optional*]

Formatting the output by adding whitespace to produce a pretty-printed, indented, human-readable form. The exact form of the transformations is not specified by this specification. Setting this feature to true will set the feature "format-canonical" to false.

`false`

[*required*] (*default*)

Don’t pretty-print the result.

IDL Definition

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

Attributes

`encoding` of type `DOMString`

The character encoding in which the output will be written.

The encoding to use when writing is determined as follows:

- If the encoding attribute has been set, that value will be used.
- If the encoding attribute is `null` or empty, but the item to be written includes an encoding declaration, that value will be used.
- If neither of the above provides an encoding name, a default encoding of "UTF-8" will be used.

The default value is `null`.

`errorHandler` of type `DOMErrorHandler`

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

`lastEncoding` of type `DOMString`, readonly

The actual character encoding that was last used by this formatter. This convenience method allows the encoding that was used when serializing a document to be directly obtained.

`newLine` of type `DOMString`

The end-of-line sequence of characters to be used in the XML being written out. The only permitted values are these:

`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 [XML] 2.11 "End-of-Line Handling".

CR

The carriage-return character (#xD).

CR-LF

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

LF

The line-feed character (#xA).

The default value for this attribute is `null`.

Methods

`canSetFeature`

Query whether setting a feature to a specific value is supported.
The feature name has the same form as a DOM hasFeature string.

Parameters

`name` of type `DOMString`

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

`state` of type `boolean`

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

Return Value

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

No Exceptions

`getFeature`

Look up the value of a feature.
The feature name has the same form as a DOM hasFeature string

Parameters

`name` of type `DOMString`

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

Return Value

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

Exceptions

`DOMException`	Raise a NOT_FOUND_ERR When the `DOMWriter` does not recognize the feature name.

`setFeature`

Set the state of a feature.

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

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

Parameters

`name` of type `DOMString`

The feature name.

`state` of type `boolean`

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

Exceptions

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

No Return Value

`writeNode`

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

Parameters

`destination` of type `DOMOutputStream`

The destination for the data to be written.

`wnode` of type `Node`

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

Return Value

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

Exceptions

`DOMSystemException`	This exception will be raised in response to any sort of IO or system error that occurs while writing to the destination. It may wrap an underlying system exception.

`writeToString`

Serialize the specified node as described above in the description of `DOMWriter`. The result of serializing the node is returned as a string. Writing a Document or Entity node produces a serialized form that is well formed XML. Writing other node types produces a fragment of text in a form that is not fully defined by this document, but that should be useful to a human for debugging or diagnostic purposes.

Parameters

`wnode` of type `Node`

The node to be written.

Return Value

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

Exceptions

`DOMException`	DOMSTRING_SIZE_ERR: The resulting string is too long to fit in a `DOMString`.

Interface *DOMInputSource*

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

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

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

There are two places that the application will deliver this input source to the parser: as the argument to the `parse` method, or as the return value of the `DOMEntityResolver.resolveEntity` [p.84] method.

The `DOMBuilder` [p.66] will use the `DOMInputSource` object to determine how to read XML input. If there is a character stream available, the parser will read that stream directly; if not, the parser will use a byte stream, if available; if neither a character stream nor a byte stream is available, the parser will attempt to open a URI connection to the resource identified by the system identifier.

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

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

IDL Definition

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

Attributes

`baseURI` of type `DOMString`

The base URI to be used (see section 5.1.4 in [RFC2396]) for resolving relative URIs to absolute URIs. If the baseURI is itself a relative URI, the behavior is implementation dependent.

`byteStream` of type `DOMInputSource` [p.81]

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

`characterStream` of type `DOMReader`

An attribute of a language-binding dependent type that represents a stream of *16-bit units.* [p.135] Application must encode the stream using UTF-16 (defined in [Unicode 3.0] and Amendment 1 of [ISO/IEC 10646]).
If a character stream is specified, the parser will ignore any byte stream and will not attempt to open a URI connection to the system identifier.

`encoding` of type `DOMString`

The character encoding, if known. The encoding must be a string acceptable for an XML encoding declaration ([XML] section 4.3.3 "Character Encoding in Entities").
This attribute has no effect when the application provides a character stream. For other sources of input, an encoding specified by means of this attribute will override any encoding specified in the XML claration or the Text Declaration, or an encoding obtained from a higher level protocol, such as HTTP [RFC2616].

`publicId` of type `DOMString`

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

`stringData` of type `DOMString`

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

`systemId` of type `DOMString`

The system identifier, a URI reference [RFC2396], for this input source. The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to fetch the ressource identifier by the URI reference only if there is no byte stream or character stream specified).

If the application knows the character encoding of the object pointed to by the system identifier, it can register the encoding by setting the encoding attribute.
If the system ID is a relative URI reference (see section 5 in [RFC2396]), the behavior is implementation dependent.

Interface *LSLoadEvent*

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

IDL Definition

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

Attributes

`inputSource` of type `DOMInputSource` [p.81] , readonly
The input source that was parsed.

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

Interface *LSProgressEvent*

This interface represents a progress event object that notifies the application about progress as a document is parsed. This event is optional and the rate at which this event is fired is implementation dependent.

IDL Definition

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

Attributes

`inputSource` of type `DOMInputSource` [p.81] , readonly
The input source that is being parsed.

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

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

Interface *DOMEntityResolver*

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

Applications needing to implement customized handling for external entities must implement this interface and register their implementation by setting the `entityResolver` property of the `DOMBuilder` [p.66] .

The `DOMBuilder` [p.66] 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 URI types other than URIs.

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

IDL Definition

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

Methods

`resolveEntity`

Allow the application to resolve external entities.
The `DOMBuilder` [p.66] will call this method before opening any external entity except the top-level document entity (including the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element); the application may request that the `DOMBuilder` resolve the entity itself, that it use an alternative URI, or that it use an entirely different input source.
Application writers can use this method to redirect external system identifiers to secure and/or local URIs, to look up public identifiers in a catalogue, or to read an entity from a database or other input source (including, for example, a dialog box).
If the system identifier is a URI, the `DOMBuilder` [p.66] must resolve it fully before reporting it to the application through this interface.
(***ED:*** See issue #4. An alternative would be to pass the URI out without resolving it, and to provide a base as an additional parameter. SAX resolves URIs first, and does not provide a base.)

Parameters

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

`systemId` of type `DOMString`
The system identifier, a URI reference [RFC2396], of the external entity being referenced exactly as written in the source (i.e. .

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

Return Value

`DOMInputSource` [p.81]	A `DOMInputSource` object describing the new input source, or `null` to request that the parser open a regular URI connection to the system identifier.

Exceptions

`DOMSystemException`	Any `DOMSystemException`, possibly wrapping another exception.

Interface *DOMBuilderFilter*

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

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

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.

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

```
interface DOMBuilderFilter {
  unsigned long      startNode(in Node snode);
  unsigned long      endNode(in Node enode);
  readonly attribute unsigned long    whatToShow;
};
```

Attributes

`whatToShow` of type `unsigned long`, readonly
: Tells the `DOMBuilder` [p.66] what types of nodes to show to the filter. See `NodeFilter` for definition of the constants. The constant `SHOW_ATTRIBUTE` is meaningless here, attribute nodes will never be passed to a `DOMBuilderFilter`.

Methods

`endNode`
: This method will be called by the parser at the completion of the parse of each node. The node will exist and be complete, as will all of its children, and their children, recursively. The parent node will also exist, although that node may be incomplete, as 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. This node may also be removed from its parent node,

which will prevent it from appearing in the final document at the completion of the parse. Aside from this one operation on the node's parent, the state of the rest of the document outside of this node is not defined, and the affect of any attempt to navigate to or 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.

Parameters

`enode` of type `Node`

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

Return Value

`unsigned long`	• `ACCEPT` if this `Node` should be included in the DOM document being built. • `REJECT` if the `Node` and all of its children should be rejected.

No Exceptions

`startNode`

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

`snode` of type `Node`

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

Issue startNode-1:

Should the parameter be an Element since we only passed elements to startNode?

Return Value

`unsigned long`	● `ACCEPT` if this `Element` should be included in the DOM document being built. ● `REJECT` if the `Element` and all of its children should be rejected. ● `SKIP` if the `Element` should be rejected. All of its children are inserted in place of the rejected `Element` node.

No Exceptions

Interface *DOMWriterFilter*

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

IDL Definition

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

Attributes

`whatToShow` of type `unsigned long`, readonly

Tells the `DOMWriter` [p.74] what types of nodes to show to the filter. See `NodeFilter` for definition of the constants. The constant `SHOW_ATTRIBUTE` is meaningless here, attribute nodes will never be passed to a `DOMWriterFilter`.

Interface *DocumentLS*

The DocumentLS interface provides a mechanism by which the content of a document can be replaced with the DOM tree produced when loading a URI, or parsing a string. The expectation is that an instance of the DocumentLS interface can be obtained by using binding-specific casting methods on an instance of the Document interface.

uses the default features.

IDL Definition

```
interface DocumentLS {
           attribute boolean          async;
  void               abort();
  boolean            load(in DOMString uri);
  boolean            loadXML(in DOMString source);
  DOMString          saveXML(in Node snode)
                                        raises(DOMException);
};
```

Attributes

`async` of type `boolean`

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

implementation doesn't support the mode the attribute is being set to.

Issue async-1:

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

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

Methods

`abort`

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

No Parameters

No Return Value

No Exceptions

`load`

Replaces the content of the document with the result of parsing the given URI. Invoking this method will either block the caller or return to the caller immediately depending on the value of the async attribute. Once the document is fully loaded the document will fire a "load" event that the caller can register as a listener for. If an error occurs the document will fire an "error" event so that the caller knows that the load failed (see `ParseErrorEvent` [p.89]).

Parameters

`uri` of type `DOMString`

The URI reference for the XML file to be loaded. If this is a relative URI...

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.

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 to a string (i.e. serialize the document or node).

Parameters

`snode` of type `Node`

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

Return Value

`DOMString`	The serialized document or `null`.

Exceptions

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

Interface *ParseErrorEvent*

ParseErrorEvent is the event that is fired if there’s an error in the XML document being parsed.

IDL Definition

```
interface ParseErrorEvent : events::Event {
  readonly attribute DOMError         error;
};
```

Attributes

`error` of type `DOMError`, readonly

An non-zero implementation dependent error code describing the error, or `0` if there is no error.

Appendix A: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] 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/2001/WD-DOM-Level-3-ASLS-20011025/idl.zip

as.idl:

```
// File: as.idl

#ifndef _AS_IDL_
#define _AS_IDL_

#include "dom.idl"
#include "ls.idl"

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

  typedef dom::DOMString DOMString;
  typedef dom::Node Node;
  typedef dom::NodeList NodeList;
  typedef dom::Attr Attr;
  typedef dom::DOMOutputStream DOMOutputStream;

  interface ASModel;
  interface ASContentModel;
  interface ASAttributeDeclaration;
  interface DOMASBuilder;
  interface DOMASWriter;

  exception DOMASException {
    unsigned short   code;
  };
  // ASExceptionCode
  const unsigned short      DUPLICATE_NAME_ERR             = 1;
  const unsigned short      TYPE_ERR                       = 2;
  const unsigned short      NO_AS_AVAILABLE                = 3;
  const unsigned short      WRONG_MIME_TYPE_ERR            = 4;


  interface ASObject {

    // ASObjectType
    const unsigned short      AS_ELEMENT_DECLARATION         = 1;
    const unsigned short      AS_ATTRIBUTE_DECLARATION       = 2;
    const unsigned short      AS_NOTATION_DECLARATION        = 3;
    const unsigned short      AS_ENTITY_DECLARATION          = 4;
    const unsigned short      AS_CONTENTMODEL                = 5;
    const unsigned short      AS_MODEL                       = 6;

    readonly attribute unsigned short   asNodeType;
             attribute ASModel          ownerASModel;
```

```
           attribute DOMString        nodeName;
           attribute DOMString        prefix;
           attribute DOMString        localName;
           attribute DOMString        namespaceURI;
  ASObject           cloneASObject(in boolean deep);
};

interface ASObjectList {
  readonly attribute unsigned long    length;
  ASObject           item(in unsigned long index);
};

interface ASNamedObjectMap {
  readonly attribute unsigned long    length;
  ASObject           getNamedItem(in DOMString name);
  ASObject           getNamedItemNS(in DOMString namespaceURI,
                                  in DOMString localName);
  ASObject           item(in unsigned long index);
  ASObject           removeNamedItem(in DOMString name)
                                    raises(dom::DOMException);
  ASObject           removeNamedItemNS(in DOMString namespaceURI,
                                     in DOMString localName)
                                    raises(dom::DOMException);
  ASObject           setNamedItem(in ASObject newASObject)
                                    raises(dom::DOMException);
  ASObject           setNamedItemNS(in ASObject newASObject)
                                    raises(dom::DOMException);
};

interface ASDataType {
  readonly attribute unsigned short   dataType;

  // DATA_TYPES
  const unsigned short      STRING_DATATYPE                = 1;
  const unsigned short      NOTATION_DATATYPE              = 10;
  const unsigned short      ID_DATATYPE                    = 11;
  const unsigned short      IDREF_DATATYPE                 = 12;
  const unsigned short      IDREFS_DATATYPE                = 13;
  const unsigned short      ENTITY_DATATYPE                = 14;
  const unsigned short      ENTITIES_DATATYPE              = 15;
  const unsigned short      NMTOKEN_DATATYPE               = 16;
  const unsigned short      NMTOKENS_DATATYPE              = 17;
  const unsigned short      BOOLEAN_DATATYPE               = 100;
  const unsigned short      FLOAT_DATATYPE                 = 101;
  const unsigned short      DOUBLE_DATATYPE                = 102;
  const unsigned short      DECIMAL_DATATYPE               = 103;
  const unsigned short      HEXBINARY_DATATYPE             = 104;
  const unsigned short      BASE64BINARY_DATATYPE          = 105;
  const unsigned short      ANYURI_DATATYPE                = 106;
  const unsigned short      QNAME_DATATYPE                 = 107;
  const unsigned short      DURATION_DATATYPE              = 108;
  const unsigned short      DATETIME_DATATYPE              = 109;
  const unsigned short      DATE_DATATYPE                  = 110;
  const unsigned short      TIME_DATATYPE                  = 111;
  const unsigned short      GYEARMONTH_DATATYPE            = 112;
  const unsigned short      GYEAR_DATATYPE                 = 113;
  const unsigned short      GMONTHDAY_DATATYPE             = 114;
  const unsigned short      GDAY_DATATYPE                  = 115;
  const unsigned short      GMONTH_DATATYPE                = 116;
  const unsigned short      INTEGER                        = 117;
  const unsigned short      NAME_DATATYPE                  = 200;
  const unsigned short      NCNAME_DATATYPE                = 201;
  const unsigned short      NORMALIZEDSTRING_DATATYPE      = 202;
  const unsigned short      TOKEN_DATATYPE                 = 203;
  const unsigned short      LANGUAGE_DATATYPE              = 204;
  const unsigned short      NONPOSITIVEINTEGER_DATATYPE    = 205;
  const unsigned short      NEGATIVEINTEGER_DATATYPE       = 206;
  const unsigned short      LONG_DATATYPE                  = 207;
  const unsigned short      INT_DATATYPE                   = 208;
  const unsigned short      SHORT_DATATYPE                 = 209;
  const unsigned short      BYTE_DATATYPE                  = 210;
  const unsigned short      NONNEGATIVEINTEGER_DATATYPE    = 211;
  const unsigned short      UNSIGNEDLONG_DATATYPE          = 212;
  const unsigned short      UNSIGNEDINT_DATATYPE           = 213;
  const unsigned short      UNSIGNEDSHORT_DATATYPE         = 214;
  const unsigned short      UNSIGNEDBYTE_DATATYPE          = 215;
  const unsigned short      POSITIVEINTEGER_DATATYPE       = 216;
  const unsigned short      OTHER_SIMPLE_DATATYPE          = 1000;
  const unsigned short      COMPLEX_DATATYPE               = 1001;
};

interface ASElementDeclaration : ASObject {

  // CONTENT_MODEL_TYPES
  const unsigned short      EMPTY_CONTENTTYPE              = 1;
  const unsigned short      ANY_CONTENTTYPE                = 2;
  const unsigned short      MIXED_CONTENTTYPE              = 3;
  const unsigned short      ELEMENTS_CONTENTTYPE           = 4;

           attribute boolean          strictMixedContent;
           attribute ASDataType       elementType;
           attribute boolean          isPCDataOnly;
           attribute unsigned short   contentType;
           attribute DOMString        systemId;
           attribute ASContentModel   asCM;
           attribute ASNamedObjectMap  ASAttributeDecls;
  void               addASAttributeDecl(in ASAttributeDeclaration attributeDecl);
  ASAttributeDeclaration removeASAttributeDecl(in ASAttributeDeclaration attributeDecl);
};

interface ASContentModel : ASObject {
  const unsigned long       AS_UNBOUNDED                   = MAX_VALUE;

  // ASContentModelType
  const unsigned short      AS_SEQUENCE                    = 0;
  const unsigned short      AS_CHOICE                      = 1;
  const unsigned short      AS_ALL                         = 2;
  const unsigned short      AS_NONE                        = 3;

           attribute unsigned short   listOperator;
           attribute unsigned long    minOccurs;
           attribute unsigned long    maxOccurs;
           attribute ASObjectList     subModels;
  void               removesubModel(in ASObject oldNode);
  void               insertsubModel(in ASObject newNode)
                                      raises(DOMASException);
  unsigned long      appendsubModel(in ASObject newNode)
                                      raises(DOMASException);
```

```
};

interface ASAttributeDeclaration : ASObject {

  // VALUE_TYPES
  const unsigned short      VALUE_NONE                     = 0;
  const unsigned short      VALUE_DEFAULT                  = 1;
  const unsigned short      VALUE_FIXED                    = 2;

           attribute ASDataType       dataType;
           attribute DOMString        dataValue;
           attribute DOMString        enumAttr;
           attribute ASObjectList     ownerElements;
           attribute unsigned short   defaultType;
};

interface ASEntityDeclaration : ASObject {

  // EntityType
  const unsigned short      INTERNAL_ENTITY                = 1;
  const unsigned short      EXTERNAL_ENTITY                = 2;

           attribute unsigned short   entityType;
           attribute DOMString        entityValue;
           attribute DOMString        systemId;
           attribute DOMString        publicId;
};

interface ASNotationDeclaration : ASObject {
           attribute DOMString        systemId;
           attribute DOMString        publicId;
};

interface DocumentAS {
           attribute ASModel          activeASModel;
           attribute ASObjectList     boundASModels;
  ASModel            getInternalAS();
  void               setInternalAS(in ASModel as);
  void               addAS(in ASModel as);
  void               removeAS(in ASModel as);
  ASElementDeclaration getElementDeclaration()
                                      raises(dom::DOMException);
  void               validate()
                                      raises(DOMASException);
};

interface DOMImplementationAS {
  ASModel            createAS(in boolean isNamespaceAware);
  DOMASBuilder       createDOMASBuilder();
  DOMASWriter        createDOMASWriter();
};

interface NodeEditAS {

  // ASCheckType
  const unsigned short      WF_CHECK                       = 1;
  const unsigned short      NS_WF_CHECK                    = 2;
  const unsigned short      PARTIAL_VALIDITY_CHECK         = 3;
  const unsigned short      STRICT_VALIDITY_CHECK          = 4;
  boolean            canInsertBefore(in Node newChild, 
                                     in Node refChild);
  boolean            canRemoveChild(in Node oldChild);
  boolean            canReplaceChild(in Node newChild, 
                                     in Node oldChild);
  boolean            canAppendChild(in Node newChild);
  boolean            isNodeValid(in boolean deep, 
                                 in unsigned short wFValidityCheckLevel)
                                      raises(DOMASException);
};

interface ElementEditAS : NodeEditAS {
  readonly attribute NodeList         definedElementTypes;
  unsigned short     contentType();
  boolean            canSetAttribute(in DOMString attrname, 
                                     in DOMString attrval);
  boolean            canSetAttributeNode(in Attr attrNode);
  boolean            canSetAttributeNS(in DOMString name, 
                                       in DOMString attrval, 
                                       in DOMString namespaceURI);
  boolean            canRemoveAttribute(in DOMString attrname);
  boolean            canRemoveAttributeNS(in DOMString attrname, 
                                          in DOMString namespaceURI);
  boolean            canRemoveAttributeNode(in Node attrNode);
  NodeList           getChildElements();
  NodeList           getParentElements();
  NodeList           getAttributeList();
  boolean            isElementDefined(in DOMString elemTypeName);
  boolean            isElementDefinedNS(in DOMString elemTypeName, 
                                        in DOMString namespaceURI, 
                                        in DOMString name);
};

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

interface ASModel : ASObject {
  readonly attribute boolean          isNamespaceAware;
  readonly attribute unsigned short   usageLocation;
           attribute DOMString        asLocation;
           attribute DOMString        asHint;
  readonly attribute ASNamedObjectMap  elementDeclarations;
  readonly attribute ASNamedObjectMap  attributeDeclarations;
  readonly attribute ASNamedObjectMap  notationDeclarations;
  readonly attribute ASNamedObjectMap  entityDeclarations;
  readonly attribute ASNamedObjectMap  contentModelDeclarations;
  void               setASModel(in ASModel abstractSchema);
```

```
    ASObjectList       getASModels();
    void               removeAS(in ASModel as);
    boolean            validate();
    ASElementDeclaration createASElementDeclaration(in DOMString namespaceURI,
                                                    in DOMString name)
                                    raises(dom::DOMException);
    ASAttributeDeclaration createASAttributeDeclaration(in DOMString namespaceURI,
                                                        in DOMString name)
                                    raises(dom::DOMException);
    ASNotationDeclaration createASNotationDeclaration(in DOMString namespaceURI,
                                                      in DOMString name,
                                                      in DOMString systemId,
                                                      in DOMString publicId)
                                    raises(dom::DOMException);
    ASEntityDeclaration createASEntityDeclaration(in DOMString name)
                                    raises(dom::DOMException);
    ASContentModel     createASContentModel(in unsigned long minOccurs,
                                        in unsigned long maxOccurs,
                                        in unsigned short operator)
                                    raises(DOMASException);
  };

  interface DocumentEditAS : NodeEditAS {
           attribute boolean          continuousValidityChecking;
  };

  interface DOMASBuilder : ls::DOMBuilder {
           attribute ASModel          abstractSchema;
    ASModel            parseASURI(in DOMString uri)
                                    raises(DOMASException,
                                           dom::DOMSystemException);
    ASModel            parseASInputSource(in ls::DOMInputSource is)
                                    raises(DOMASException,
                                           dom::DOMSystemException);
  };

  interface DOMASWriter : ls::DOMWriter {
    void               writeASModel(in DOMOutputStream destination,
                                    in ASModel model)
                                    raises(dom::DOMSystemException);
  };
};

#endif // _AS_IDL_
```

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 dom::DOMErrorHandler DOMErrorHandler;
  typedef dom::DOMString DOMString;
  typedef dom::Node Node;
  typedef dom::Document Document;
  typedef dom::DOMOutputStream DOMOutputStream;
  typedef dom::DOMReader DOMReader;
  typedef dom::DOMError DOMError;

  interface DOMBuilder;
  interface DOMWriter;
  interface DOMInputSource;
  interface DOMEntityResolver;
  interface DOMBuilderFilter;

  interface DOMImplementationLS {

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

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

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

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

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

  interface DOMWriter {
```

```
  void               setFeature(in DOMString name,
                                in boolean state)
                                   raises(dom::DOMException);
  boolean            canSetFeature(in DOMString name,
                                   in boolean state);
  boolean            getFeature(in DOMString name)
                                   raises(dom::DOMException);
           attribute DOMString        encoding;
  readonly attribute DOMString        lastEncoding;
           attribute DOMString        newLine;
           attribute DOMErrorHandler  errorHandler;
  boolean            writeNode(in DOMOutputStream destination,
                               in Node wnode)
                                   raises(dom::DOMSystemException);
  DOMString          writeToString(in Node wnode)
                                   raises(dom::DOMException);
};

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

interface DOMEntityResolver {
  DOMInputSource     resolveEntity(in DOMString publicId,
                                   in DOMString systemId,
                                   in DOMString baseURI)
                                   raises(dom::DOMSystemException);
};

interface DOMBuilderFilter {
  unsigned long      startNode(in Node snode);
  unsigned long      endNode(in Node enode);
  readonly attribute unsigned long    whatToShow;
};

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

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

interface LSProgressEvent : events::Event {
  readonly attribute DOMInputSource   inputSource;
```

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

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

  interface ParseErrorEvent : events::Event {
    readonly attribute DOMError         error;
  };
};

#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 Abstract Schemas and Load and Save.

The Java files are also available as
http://www.w3.org/TR/2001/WD-DOM-Level-3-ASLS-20011025/java-binding.zip

org/w3c/dom/as/DOMASException.java:

```
package org.w3c.dom.as;

public class DOMASException extends RuntimeException {
    public DOMASException(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // ASExceptionCode
    public static final short DUPLICATE_NAME_ERR        = 1;
    public static final short TYPE_ERR                  = 2;
    public static final short NO_AS_AVAILABLE           = 3;
    public static final short WRONG_MIME_TYPE_ERR       = 4;

}
```

org/w3c/dom/as/ASModel.java:

```
package org.w3c.dom.as;

import org.w3c.dom.DOMException;

public interface ASModel extends ASObject {
    public boolean getIsNamespaceAware();

    public short getUsageLocation();

    public String getAsLocation();
    public void setAsLocation(String asLocation);

    public String getAsHint();
    public void setAsHint(String asHint);

    public ASNamedObjectMap getElementDeclarations();

    public ASNamedObjectMap getAttributeDeclarations();

    public ASNamedObjectMap getNotationDeclarations();

    public ASNamedObjectMap getEntityDeclarations();

    public ASNamedObjectMap getContentModelDeclarations();

    public void setASModel(ASModel abstractSchema);
    public ASObjectList getASModels();

    public void removeAS(ASModel as);

    public boolean validate();

    public ASElementDeclaration createASElementDeclaration(String namespaceURI,
                                                           String name)
                                                           throws DOMException;

    public ASAttributeDeclaration createASAttributeDeclaration(String namespaceURI,
                                                               String name)
                                                               throws DOMException;

    public ASNotationDeclaration createASNotationDeclaration(String namespaceURI,
                                                             String name,
                                                             String systemId,
                                                             String publicId)
                                                             throws DOMException;

    public ASEntityDeclaration createASEntityDeclaration(String name)
                                                         throws DOMException;

    public ASContentModel createASContentModel(int minOccurs,
                                               int maxOccurs,
                                               short operator)
                                               throws DOMASException;

}
```

org/w3c/dom/as/ASObject.java:

```
package org.w3c.dom.as;

public interface ASObject {
    // ASObjectType
    public static final short AS_ELEMENT_DECLARATION    = 1;
    public static final short AS_ATTRIBUTE_DECLARATION  = 2;
    public static final short AS_NOTATION_DECLARATION   = 3;
    public static final short AS_ENTITY_DECLARATION     = 4;
    public static final short AS_CONTENTMODEL           = 5;
    public static final short AS_MODEL                  = 6;

    public short getAsNodeType();

    public ASModel getOwnerASModel();
    public void setOwnerASModel(ASModel ownerASModel);

    public String getNodeName();
    public void setNodeName(String nodeName);

    public String getPrefix();
    public void setPrefix(String prefix);

    public String getLocalName();
    public void setLocalName(String localName);

    public String getNamespaceURI();
    public void setNamespaceURI(String namespaceURI);

    public ASObject cloneASObject(boolean deep);

}
```

org/w3c/dom/as/ASObjectList.java:

```
package org.w3c.dom.as;

public interface ASObjectList {
    public int getLength();

    public ASObject item(int index);

}
```

org/w3c/dom/as/ASNamedObjectMap.java:

```
package org.w3c.dom.as;

import org.w3c.dom.DOMException;

public interface ASNamedObjectMap {
    public int getLength();

    public ASObject getNamedItem(String name);

    public ASObject getNamedItemNS(String namespaceURI,
                                   String localName);

    public ASObject item(int index);

    public ASObject removeNamedItem(String name)
                                    throws DOMException;

    public ASObject removeNamedItemNS(String namespaceURI,
                                      String localName)
                                      throws DOMException;

    public ASObject setNamedItem(ASObject newASObject)
                                 throws DOMException;

    public ASObject setNamedItemNS(ASObject newASObject)
                                   throws DOMException;

}
```

org/w3c/dom/as/ASDataType.java:

```
package org.w3c.dom.as;

public interface ASDataType {
    public short getDataType();

    // DATA_TYPES
    public static final short STRING_DATATYPE           = 1;
    public static final short NOTATION_DATATYPE         = 10;
    public static final short ID_DATATYPE               = 11;
    public static final short IDREF_DATATYPE            = 12;
    public static final short IDREFS_DATATYPE           = 13;
    public static final short ENTITY_DATATYPE           = 14;
    public static final short ENTITIES_DATATYPE         = 15;
    public static final short NMTOKEN_DATATYPE          = 16;
    public static final short NMTOKENS_DATATYPE         = 17;
    public static final short BOOLEAN_DATATYPE          = 100;
    public static final short FLOAT_DATATYPE            = 101;
    public static final short DOUBLE_DATATYPE           = 102;
    public static final short DECIMAL_DATATYPE          = 103;
    public static final short HEXBINARY_DATATYPE        = 104;
    public static final short BASE64BINARY_DATATYPE     = 105;
    public static final short ANYURI_DATATYPE           = 106;
    public static final short QNAME_DATATYPE            = 107;
    public static final short DURATION_DATATYPE         = 108;
    public static final short DATETIME_DATATYPE         = 109;
    public static final short DATE_DATATYPE             = 110;
    public static final short TIME_DATATYPE             = 111;
    public static final short GYEARMONTH_DATATYPE       = 112;
    public static final short GYEAR_DATATYPE            = 113;
    public static final short GMONTHDAY_DATATYPE        = 114;
    public static final short GDAY_DATATYPE             = 115;
    public static final short GMONTH_DATATYPE           = 116;
    public static final short INTEGER                   = 117;
    public static final short NAME_DATATYPE             = 200;
    public static final short NCNAME_DATATYPE           = 201;
    public static final short NORMALIZEDSTRING_DATATYPE = 202;
    public static final short TOKEN_DATATYPE            = 203;
    public static final short LANGUAGE_DATATYPE         = 204;
    public static final short NONPOSITIVEINTEGER_DATATYPE = 205;
    public static final short NEGATIVEINTEGER_DATATYPE  = 206;
    public static final short LONG_DATATYPE             = 207;
    public static final short INT_DATATYPE              = 208;
    public static final short SHORT_DATATYPE            = 209;
    public static final short BYTE_DATATYPE             = 210;
    public static final short NONNEGATIVEINTEGER_DATATYPE = 211;
    public static final short UNSIGNEDLONG_DATATYPE     = 212;
    public static final short UNSIGNEDINT_DATATYPE      = 213;
    public static final short UNSIGNEDSHORT_DATATYPE    = 214;
    public static final short UNSIGNEDBYTE_DATATYPE     = 215;
    public static final short POSITIVEINTEGER_DATATYPE  = 216;
    public static final short OTHER_SIMPLE_DATATYPE     = 1000;
    public static final short COMPLEX_DATATYPE          = 1001;

}
```

org/w3c/dom/as/ASElementDeclaration.java:

```java
package org.w3c.dom.as;

public interface ASElementDeclaration extends ASObject {
    // CONTENT_MODEL_TYPES
    public static final short EMPTY_CONTENTTYPE         = 1;
    public static final short ANY_CONTENTTYPE           = 2;
    public static final short MIXED_CONTENTTYPE         = 3;
    public static final short ELEMENTS_CONTENTTYPE      = 4;

    public boolean getStrictMixedContent();
    public void setStrictMixedContent(boolean strictMixedContent);

    public ASDataType getElementType();
    public void setElementType(ASDataType elementType);

    public boolean getIsPCDataOnly();
    public void setIsPCDataOnly(boolean isPCDataOnly);

    public short getContentType();
    public void setContentType(short contentType);

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

    public ASContentModel getAsCM();
    public void setAsCM(ASContentModel asCM);

    public ASNamedObjectMap getASAttributeDecls();
    public void setASAttributeDecls(ASNamedObjectMap ASAttributeDecls);

    public void addASAttributeDecl(ASAttributeDeclaration attributeDecl);

    public ASAttributeDeclaration removeASAttributeDecl(ASAttributeDeclaration attributeDecl);

}
```

org/w3c/dom/as/ASContentModel.java:

```java
package org.w3c.dom.as;

public interface ASContentModel extends ASObject {
    public static final int AS_UNBOUNDED              = MAX_VALUE;
    // ASContentModelType
    public static final short AS_SEQUENCE               = 0;
    public static final short AS_CHOICE                 = 1;
    public static final short AS_ALL                    = 2;
    public static final short AS_NONE                   = 3;

    public short getListOperator();
    public void setListOperator(short listOperator);

    public int getMinOccurs();
    public void setMinOccurs(int minOccurs);

    public int getMaxOccurs();
    public void setMaxOccurs(int maxOccurs);
    public ASObjectList getSubModels();
    public void setSubModels(ASObjectList subModels);

    public void removesubModel(ASObject oldNode);

    public void insertsubModel(ASObject newNode)
                                throws DOMASException;

    public int appendsubModel(ASObject newNode)
                               throws DOMASException;

}
```

org/w3c/dom/as/ASAttributeDeclaration.java:

```
package org.w3c.dom.as;

public interface ASAttributeDeclaration extends ASObject {
    // VALUE_TYPES
    public static final short VALUE_NONE                = 0;
    public static final short VALUE_DEFAULT             = 1;
    public static final short VALUE_FIXED               = 2;

    public ASDataType getDataType();
    public void setDataType(ASDataType dataType);

    public String getDataValue();
    public void setDataValue(String dataValue);

    public String getEnumAttr();
    public void setEnumAttr(String enumAttr);

    public ASObjectList getOwnerElements();
    public void setOwnerElements(ASObjectList ownerElements);

    public short getDefaultType();
    public void setDefaultType(short defaultType);

}
```

org/w3c/dom/as/ASEntityDeclaration.java:

```
package org.w3c.dom.as;

public interface ASEntityDeclaration extends ASObject {
    // EntityType
    public static final short INTERNAL_ENTITY           = 1;
    public static final short EXTERNAL_ENTITY           = 2;

    public short getEntityType();
    public void setEntityType(short entityType);

    public String getEntityValue();
    public void setEntityValue(String entityValue);
    public String getSystemId();
    public void setSystemId(String systemId);

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

}
```

org/w3c/dom/as/ASNotationDeclaration.java:

```
package org.w3c.dom.as;

public interface ASNotationDeclaration extends ASObject {
    public String getSystemId();
    public void setSystemId(String systemId);

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

}
```

org/w3c/dom/as/DocumentAS.java:

```
package org.w3c.dom.as;

import org.w3c.dom.DOMException;

public interface DocumentAS {
    public ASModel getActiveASModel();
    public void setActiveASModel(ASModel activeASModel);

    public ASObjectList getBoundASModels();
    public void setBoundASModels(ASObjectList boundASModels);

    public ASModel getInternalAS();

    public void setInternalAS(ASModel as);

    public void addAS(ASModel as);

    public void removeAS(ASModel as);

    public ASElementDeclaration getElementDeclaration()
                                                  throws DOMException;

    public void validate()
                         throws DOMASException;

}
```

org/w3c/dom/as/DOMImplementationAS.java:

```
package org.w3c.dom.as;

public interface DOMImplementationAS {
    public ASModel createAS(boolean isNamespaceAware);

    public DOMASBuilder createDOMASBuilder();

    public DOMASWriter createDOMASWriter();

}
```

org/w3c/dom/as/DocumentEditAS.java:

```
package org.w3c.dom.as;

public interface DocumentEditAS extends NodeEditAS {
    public boolean getContinuousValidityChecking();
    public void setContinuousValidityChecking(boolean continuousValidityChecking);

}
```

org/w3c/dom/as/NodeEditAS.java:

```
package org.w3c.dom.as;

import org.w3c.dom.Node;

public interface NodeEditAS {
    // ASCheckType
    public static final short WF_CHECK                  = 1;
    public static final short NS_WF_CHECK               = 2;
    public static final short PARTIAL_VALIDITY_CHECK    = 3;
    public static final short STRICT_VALIDITY_CHECK     = 4;

    public boolean canInsertBefore(Node newChild,
                                   Node refChild);

    public boolean canRemoveChild(Node oldChild);

    public boolean canReplaceChild(Node newChild,
                                   Node oldChild);

    public boolean canAppendChild(Node newChild);

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

}
```

org/w3c/dom/as/ElementEditAS.java:

```
package org.w3c.dom.as;

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

public interface ElementEditAS extends NodeEditAS {
    public NodeList getDefinedElementTypes();

    public short contentType();

    public boolean canSetAttribute(String attrname,
                                   String attrval);

    public boolean canSetAttributeNode(Attr attrNode);

    public boolean canSetAttributeNS(String name,
                                     String attrval,
                                     String namespaceURI);

    public boolean canRemoveAttribute(String attrname);

    public boolean canRemoveAttributeNS(String attrname,
                                        String namespaceURI);

    public boolean canRemoveAttributeNode(Node attrNode);

    public NodeList getChildElements();

    public NodeList getParentElements();

    public NodeList getAttributeList();

    public boolean isElementDefined(String elemTypeName);

    public boolean isElementDefinedNS(String elemTypeName,
                                      String namespaceURI,
                                      String name);

}
```

org/w3c/dom/as/CharacterDataEditAS.java:

```
package org.w3c.dom.as;

public interface CharacterDataEditAS extends NodeEditAS {
    public boolean getIsWhitespaceOnly();

    public boolean canSetData(int offset,
                              int count);

    public boolean canAppendData(String arg);

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

    public boolean canInsertData(int offset,
                                 String arg);

    public boolean canDeleteData(int offset,
                                 int count);

}
```

org/w3c/dom/as/DOMASBuilder.java:

```
package org.w3c.dom.as;

import org.w3c.dom.ls.DOMInputSource;
import org.w3c.dom.ls.DOMBuilder;

public interface DOMASBuilder extends DOMBuilder {
    public ASModel getAbstractSchema();
    public void setAbstractSchema(ASModel abstractSchema);

    public ASModel parseASURI(String uri)
                              throws DOMASException, Exception;

    public ASModel parseASInputSource(DOMInputSource is)
                                      throws DOMASException, Exception;

}
```

org/w3c/dom/as/DOMASWriter.java:

```
package org.w3c.dom.as;

import org.w3c.dom.ls.DOMWriter;

public interface DOMASWriter extends DOMWriter {
    public void writeASModel(java.io.OutputStream destination,
                             ASModel model)
                             throws Exception;

}
```

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

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

import org.w3c.dom.DOMException;

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

    public DOMBuilder createDOMBuilder(short mode)
```

```
                                        throws DOMException;

    public DOMWriter createDOMWriter();

    public DOMInputSource createDOMInputSource();

}
```

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

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

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

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

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

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

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

    public boolean canSetFeature(String name,
                                 boolean state);

    public boolean getFeature(String name)
                              throws DOMException;

    public Document parseURI(String uri)
                             throws Exception;

    public Document parse(DOMInputSource is)
                          throws Exception;

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

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

}
```

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

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

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

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

    public boolean canSetFeature(String name,
                                 boolean state);

    public boolean getFeature(String name)
                              throws DOMException;

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

    public String getLastEncoding();

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

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

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

    public String writeToString(Node wnode)
                                throws DOMException;

}
```

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

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

public interface DOMInputSource {
    public DOMInputSource getByteStream();
    public void setByteStream(DOMInputSource byteStream);

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

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

    public String getEncoding();
    public void setEncoding(String encoding);
    public String getPublicId();
    public void setPublicId(String publicId);

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

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

}
```

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

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

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

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

    public DOMInputSource getInputSource();

}
```

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

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

import org.w3c.dom.events.Event;

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

    public int getPosition();

    public int getTotalSize();

}
```

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

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

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

}
```

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

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

import org.w3c.dom.Node;

public interface DOMBuilderFilter {
    public int startNode(Node snode);

    public int endNode(Node enode);

    public int getWhatToShow();

}
```

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

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

import org.w3c.dom.traversal.NodeFilter;

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

}
```

org/w3c/dom/ls/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);

    public void abort();

    public boolean load(String uri);

    public boolean loadXML(String source);

    public String saveXML(Node snode)
                          throws DOMException;

}
```

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

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

import org.w3c.dom.events.Event;
import org.w3c.dom.DOMError;
```

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

```
public interface ParseErrorEvent extends Event {
    public DOMError getError();

}
```

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

Appendix C: ECMAScript Language Binding

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

Prototype Object **DOMASException**

The **DOMASException** class has the following constants:

DOMASException.DUPLICATE_NAME_ERR

This constant is of type **Number** and its value is **1**.

DOMASException.TYPE_ERR

This constant is of type **Number** and its value is **2**.

DOMASException.NO_AS_AVAILABLE

This constant is of type **Number** and its value is **3**.

DOMASException.WRONG_MIME_TYPE_ERR

This constant is of type **Number** and its value is **4**.

Object **DOMASException**

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

code

This property is of type **Number**.

Object **ASModel**

ASModel has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.

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

isNamespaceAware

This read-only property is of type **Boolean**.

usageLocation

This read-only property is of type **Number**.

asLocation

This property is of type **String**.

asHint

This property is of type **String**.

elementDeclarations

This read-only property is a **ASNamedObjectMap** object.

attributeDeclarations

This read-only property is a **ASNamedObjectMap** object.

notationDeclarations

This read-only property is a **ASNamedObjectMap** object.

entityDeclarations

This read-only property is a **ASNamedObjectMap** object.

contentModelDeclarations

This read-only property is a **ASNamedObjectMap** object.

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

setASModel(abstractSchema)

This method has no return value.

The **abstractSchema** parameter is a **ASModel** object.

getASModels()
This method returns a **ASObjectList** object.
removeAS(as)
This method has no return value.
The **as** parameter is a **ASModel** object.
validate()
This method returns a **Boolean**.
createASElementDeclaration(namespaceURI, name)
This method returns a **ASElementDeclaration** object.
The **namespaceURI** parameter is of type **String**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.
createASAttributeDeclaration(namespaceURI, name)
This method returns a **ASAttributeDeclaration** object.
The **namespaceURI** parameter is of type **String**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.
createASNotationDeclaration(namespaceURI, name, systemId, publicId)
This method returns a **ASNotationDeclaration** object.
The **namespaceURI** parameter is of type **String**.
The **name** parameter is of type **String**.
The **systemId** parameter is of type **String**.
The **publicId** parameter is of type **String**.
This method can raise a **DOMException** object.
createASEntityDeclaration(name)
This method returns a **ASEntityDeclaration** object.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.
createASContentModel(minOccurs, maxOccurs, operator)
This method returns a **ASContentModel** object.
The **minOccurs** parameter is of type **Number**.
The **maxOccurs** parameter is of type **Number**.
The **operator** parameter is of type **Number**.
This method can raise a **DOMASException** object.

Prototype Object **ASObject**
The **ASObject** class has the following constants:
ASObject.AS_ELEMENT_DECLARATION
This constant is of type **Number** and its value is **1**.
ASObject.AS_ATTRIBUTE_DECLARATION
This constant is of type **Number** and its value is **2**.
ASObject.AS_NOTATION_DECLARATION
This constant is of type **Number** and its value is **3**.
ASObject.AS_ENTITY_DECLARATION
This constant is of type **Number** and its value is **4**.
ASObject.AS_CONTENTMODEL
This constant is of type **Number** and its value is **5**.

ASObject.AS_MODEL
This constant is of type **Number** and its value is **6**.

Object **ASObject**

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

asNodeType
This read-only property is of type **Number**.

ownerASModel
This property is a **ASModel** object.

nodeName
This property is of type **String**.

prefix
This property is of type **String**.

localName
This property is of type **String**.

namespaceURI
This property is of type **String**.

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

cloneASObject(deep)
This method returns a **ASObject** object.
The **deep** parameter is of type **Boolean**.

Object **ASObjectList**

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

length
This read-only property is of type **Number**.

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

item(index)
This method returns a **ASObject** object.
The **index** parameter is of type **Number**.
Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

Object **ASNamedObjectMap**

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

length
This read-only property is of type **Number**.

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

getNamedItem(name)
This method returns a **ASObject** object.
The **name** parameter is of type **String**.

getNamedItemNS(namespaceURI, localName)
This method returns a **ASObject** object.
The **namespaceURI** parameter is of type **String**.
The **localName** parameter is of type **String**.

item(index)
This method returns a **ASObject** object.
The **index** parameter is of type **Number**.

Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

removeNamedItem(name)
This method returns a **ASObject** object.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

removeNamedItemNS(namespaceURI, localName)
This method returns a **ASObject** object.
The **namespaceURI** parameter is of type **String**.
The **localName** parameter is of type **String**.
This method can raise a **DOMException** object.

setNamedItem(newASObject)
This method returns a **ASObject** object.
The **newASObject** parameter is a **ASObject** object.
This method can raise a **DOMException** object.

setNamedItemNS(newASObject)
This method returns a **ASObject** object.
The **newASObject** parameter is a **ASObject** object.
This method can raise a **DOMException** object.

Prototype Object **ASDataType**

The **ASDataType** class has the following constants:

ASDataType.STRING_DATATYPE
This constant is of type **Number** and its value is **1**.

ASDataType.NOTATION_DATATYPE
This constant is of type **Number** and its value is **10**.

ASDataType.ID_DATATYPE
This constant is of type **Number** and its value is **11**.

ASDataType.IDREF_DATATYPE
This constant is of type **Number** and its value is **12**.

ASDataType.IDREFS_DATATYPE
This constant is of type **Number** and its value is **13**.

ASDataType.ENTITY_DATATYPE
This constant is of type **Number** and its value is **14**.

ASDataType.ENTITIES_DATATYPE
This constant is of type **Number** and its value is **15**.

ASDataType.NMTOKEN_DATATYPE
This constant is of type **Number** and its value is **16**.

ASDataType.NMTOKENS_DATATYPE
This constant is of type **Number** and its value is **17**.

ASDataType.BOOLEAN_DATATYPE
This constant is of type **Number** and its value is **100**.

ASDataType.FLOAT_DATATYPE
This constant is of type **Number** and its value is **101**.

ASDataType.DOUBLE_DATATYPE
This constant is of type **Number** and its value is **102**.

ASDataType.DECIMAL_DATATYPE
This constant is of type **Number** and its value is **103**.
ASDataType.HEXBINARY_DATATYPE
This constant is of type **Number** and its value is **104**.
ASDataType.BASE64BINARY_DATATYPE
This constant is of type **Number** and its value is **105**.
ASDataType.ANYURI_DATATYPE
This constant is of type **Number** and its value is **106**.
ASDataType.QNAME_DATATYPE
This constant is of type **Number** and its value is **107**.
ASDataType.DURATION_DATATYPE
This constant is of type **Number** and its value is **108**.
ASDataType.DATETIME_DATATYPE
This constant is of type **Number** and its value is **109**.
ASDataType.DATE_DATATYPE
This constant is of type **Number** and its value is **110**.
ASDataType.TIME_DATATYPE
This constant is of type **Number** and its value is **111**.
ASDataType.GYEARMONTH_DATATYPE
This constant is of type **Number** and its value is **112**.
ASDataType.GYEAR_DATATYPE
This constant is of type **Number** and its value is **113**.
ASDataType.GMONTHDAY_DATATYPE
This constant is of type **Number** and its value is **114**.
ASDataType.GDAY_DATATYPE
This constant is of type **Number** and its value is **115**.
ASDataType.GMONTH_DATATYPE
This constant is of type **Number** and its value is **116**.
ASDataType.INTEGER
This constant is of type **Number** and its value is **117**.
ASDataType.NAME_DATATYPE
This constant is of type **Number** and its value is **200**.
ASDataType.NCNAME_DATATYPE
This constant is of type **Number** and its value is **201**.
ASDataType.NORMALIZEDSTRING_DATATYPE
This constant is of type **Number** and its value is **202**.
ASDataType.TOKEN_DATATYPE
This constant is of type **Number** and its value is **203**.
ASDataType.LANGUAGE_DATATYPE
This constant is of type **Number** and its value is **204**.
ASDataType.NONPOSITIVEINTEGER_DATATYPE
This constant is of type **Number** and its value is **205**.
ASDataType.NEGATIVEINTEGER_DATATYPE
This constant is of type **Number** and its value is **206**.
ASDataType.LONG_DATATYPE
This constant is of type **Number** and its value is **207**.

ASDataType.INT_DATATYPE
This constant is of type **Number** and its value is **208**.
ASDataType.SHORT_DATATYPE
This constant is of type **Number** and its value is **209**.
ASDataType.BYTE_DATATYPE
This constant is of type **Number** and its value is **210**.
ASDataType.NONNEGATIVEINTEGER_DATATYPE
This constant is of type **Number** and its value is **211**.
ASDataType.UNSIGNEDLONG_DATATYPE
This constant is of type **Number** and its value is **212**.
ASDataType.UNSIGNEDINT_DATATYPE
This constant is of type **Number** and its value is **213**.
ASDataType.UNSIGNEDSHORT_DATATYPE
This constant is of type **Number** and its value is **214**.
ASDataType.UNSIGNEDBYTE_DATATYPE
This constant is of type **Number** and its value is **215**.
ASDataType.POSITIVEINTEGER_DATATYPE
This constant is of type **Number** and its value is **216**.
ASDataType.OTHER_SIMPLE_DATATYPE
This constant is of type **Number** and its value is **1000**.
ASDataType.COMPLEX_DATATYPE
This constant is of type **Number** and its value is **1001**.

Object **ASDataType**
The **ASDataType** object has the following properties:
dataType
This read-only property is of type **Number**.

Prototype Object **ASElementDeclaration**
The **ASElementDeclaration** class has the following constants:
ASElementDeclaration.EMPTY_CONTENTTYPE
This constant is of type **Number** and its value is **1**.
ASElementDeclaration.ANY_CONTENTTYPE
This constant is of type **Number** and its value is **2**.
ASElementDeclaration.MIXED_CONTENTTYPE
This constant is of type **Number** and its value is **3**.
ASElementDeclaration.ELEMENTS_CONTENTTYPE
This constant is of type **Number** and its value is **4**.

Object **ASElementDeclaration**
ASElementDeclaration has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.
The **ASElementDeclaration** object has the following properties:
strictMixedContent
This property is of type **Boolean**.
elementType
This property is a **ASDataType** object.
isPCDataOnly
This property is of type **Boolean**.

contentType
This property is of type **Number**.
systemId
This property is of type **String**.
asCM
This property is a **ASContentModel** object.
ASAttributeDecls
This property is a **ASNamedObjectMap** object.
The **ASElementDeclaration** object has the following methods:
addASAttributeDecl(attributeDecl)
This method has no return value.
The **attributeDecl** parameter is a **ASAttributeDeclaration** object.
removeASAttributeDecl(attributeDecl)
This method returns a **ASAttributeDeclaration** object.
The **attributeDecl** parameter is a **ASAttributeDeclaration** object.
Prototype Object **ASContentModel**
The **ASContentModel** class has the following constants:
ASContentModel.AS_UNBOUNDED
This constant is of type **Number** and its value is **MAX_VALUE**.
ASContentModel.AS_SEQUENCE
This constant is of type **Number** and its value is **0**.
ASContentModel.AS_CHOICE
This constant is of type **Number** and its value is **1**.
ASContentModel.AS_ALL
This constant is of type **Number** and its value is **2**.
ASContentModel.AS_NONE
This constant is of type **Number** and its value is **3**.
Object **ASContentModel**
ASContentModel has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.
The **ASContentModel** object has the following properties:
listOperator
This property is of type **Number**.
minOccurs
This property is of type **Number**.
maxOccurs
This property is of type **Number**.
subModels
This property is a **ASObjectList** object.
The **ASContentModel** object has the following methods:
removesubModel(oldNode)
This method has no return value.
The **oldNode** parameter is a **ASObject** object.
insertsubModel(newNode)
This method has no return value.
The **newNode** parameter is a **ASObject** object.

This method can raise a **DOMASException** object.

appendsubModel(newNode)

This method returns a **Number**.

The **newNode** parameter is a **ASObject** object.

This method can raise a **DOMASException** object.

Prototype Object **ASAttributeDeclaration**

The **ASAttributeDeclaration** class has the following constants:

ASAttributeDeclaration.VALUE_NONE

This constant is of type **Number** and its value is **0**.

ASAttributeDeclaration.VALUE_DEFAULT

This constant is of type **Number** and its value is **1**.

ASAttributeDeclaration.VALUE_FIXED

This constant is of type **Number** and its value is **2**.

Object **ASAttributeDeclaration**

ASAttributeDeclaration has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.

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

dataType

This property is a **ASDataType** object.

dataValue

This property is of type **String**.

enumAttr

This property is of type **String**.

ownerElements

This property is a **ASObjectList** object.

defaultType

This property is of type **Number**.

Prototype Object **ASEntityDeclaration**

The **ASEntityDeclaration** class has the following constants:

ASEntityDeclaration.INTERNAL_ENTITY

This constant is of type **Number** and its value is **1**.

ASEntityDeclaration.EXTERNAL_ENTITY

This constant is of type **Number** and its value is **2**.

Object **ASEntityDeclaration**

ASEntityDeclaration has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.

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

entityType

This property is of type **Number**.

entityValue

This property is of type **String**.

systemId

This property is of type **String**.

publicId

This property is of type **String**.

Object **ASNotationDeclaration**

ASNotationDeclaration has the all the properties and methods of the **ASObject** object as well as the properties and methods defined below.

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

systemId

This property is of type **String**.

publicId

This property is of type **String**.

Object **DocumentAS**

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

activeASModel

This property is a **ASModel** object.

boundASModels

This property is a **ASObjectList** object.

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

getInternalAS()

This method returns a **ASModel** object.

setInternalAS(as)

This method has no return value.

The **as** parameter is a **ASModel** object.

addAS(as)

This method has no return value.

The **as** parameter is a **ASModel** object.

removeAS(as)

This method has no return value.

The **as** parameter is a **ASModel** object.

getElementDeclaration()

This method returns a **ASElementDeclaration** object.

This method can raise a **DOMException** object.

validate()

This method has no return value.

This method can raise a **DOMASException** object.

Object **DOMImplementationAS**

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

createAS(isNamespaceAware)

This method returns a **ASModel** object.

The **isNamespaceAware** parameter is of type **Boolean**.

createDOMASBuilder()

This method returns a **DOMASBuilder** object.

createDOMASWriter()

This method returns a **DOMASWriter** object.

Object **DocumentEditAS**

DocumentEditAS has the all the properties and methods of the **NodeEditAS** object as well as the properties and methods defined below.

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

continuousValidityChecking
This property is of type **Boolean**.

Prototype Object **NodeEditAS**

The **NodeEditAS** class has the following constants:

NodeEditAS.WF_CHECK
This constant is of type **Number** and its value is **1**.

NodeEditAS.NS_WF_CHECK
This constant is of type **Number** and its value is **2**.

NodeEditAS.PARTIAL_VALIDITY_CHECK
This constant is of type **Number** and its value is **3**.

NodeEditAS.STRICT_VALIDITY_CHECK
This constant is of type **Number** and its value is **4**.

Object **NodeEditAS**

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

canInsertBefore(newChild, refChild)
This method returns a **Boolean**.
The **newChild** parameter is a **Node** object.
The **refChild** parameter is a **Node** object.

canRemoveChild(oldChild)
This method returns a **Boolean**.
The **oldChild** parameter is a **Node** object.

canReplaceChild(newChild, oldChild)
This method returns a **Boolean**.
The **newChild** parameter is a **Node** object.
The **oldChild** parameter is a **Node** object.

canAppendChild(newChild)
This method returns a **Boolean**.
The **newChild** parameter is a **Node** object.

isNodeValid(deep, wFValidityCheckLevel)
This method returns a **Boolean**.
The **deep** parameter is of type **Boolean**.
The **wFValidityCheckLevel** parameter is of type **Number**.
This method can raise a **DOMASException** object.

Object **ElementEditAS**

ElementEditAS has the all the properties and methods of the **NodeEditAS** object as well as the properties and methods defined below.

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

definedElementTypes
This read-only property is a **NodeList** object.

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

contentType()
This method returns a **Number**.

canSetAttribute(attrname, attrval)
This method returns a **Boolean**.
The **attrname** parameter is of type **String**.
The **attrval** parameter is of type **String**.

canSetAttributeNode(attrNode)
This method returns a **Boolean**.
The **attrNode** parameter is a **Attr** object.

canSetAttributeNS(name, attrval, namespaceURI)
This method returns a **Boolean**.
The **name** parameter is of type **String**.
The **attrval** parameter is of type **String**.
The **namespaceURI** parameter is of type **String**.

canRemoveAttribute(attrname)
This method returns a **Boolean**.
The **attrname** parameter is of type **String**.

canRemoveAttributeNS(attrname, namespaceURI)
This method returns a **Boolean**.
The **attrname** parameter is of type **String**.
The **namespaceURI** parameter is of type **String**.

canRemoveAttributeNode(attrNode)
This method returns a **Boolean**.
The **attrNode** parameter is a **Node** object.

getChildElements()
This method returns a **NodeList** object.

getParentElements()
This method returns a **NodeList** object.

getAttributeList()
This method returns a **NodeList** object.

isElementDefined(elemTypeName)
This method returns a **Boolean**.
The **elemTypeName** parameter is of type **String**.

isElementDefinedNS(elemTypeName, namespaceURI, name)
This method returns a **Boolean**.
The **elemTypeName** parameter is of type **String**.
The **namespaceURI** parameter is of type **String**.
The **name** parameter is of type **String**.

Object **CharacterDataEditAS**

CharacterDataEditAS has the all the properties and methods of the **NodeEditAS** object as well as the properties and methods defined below.

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

isWhitespaceOnly
This read-only property is of type **Boolean**.

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

canSetData(offset, count)
This method returns a **Boolean**.
The **offset** parameter is of type **Number**.
The **count** parameter is of type **Number**.

canAppendData(arg)
This method returns a **Boolean**.
The **arg** parameter is of type **String**.

canReplaceData(offset, count, arg)
This method returns a **Boolean**.
The **offset** parameter is of type **Number**.
The **count** parameter is of type **Number**.
The **arg** parameter is of type **String**.
canInsertData(offset, arg)
This method returns a **Boolean**.
The **offset** parameter is of type **Number**.
The **arg** parameter is of type **String**.
canDeleteData(offset, count)
This method returns a **Boolean**.
The **offset** parameter is of type **Number**.
The **count** parameter is of type **Number**.

Object **DOMASBuilder**
DOMASBuilder has the all the properties and methods of the **DOMBuilder** object as well as the properties and methods defined below.
The **DOMASBuilder** object has the following properties:
abstractSchema
This property is a **ASModel** object.
The **DOMASBuilder** object has the following methods:
parseASURI(uri)
This method returns a **ASModel** object.
The **uri** parameter is of type **String**.
This method can raise a **DOMASException** object or a **DOMSystemException** object.
parseASInputSource(is)
This method returns a **ASModel** object.
The **is** parameter is a **DOMInputSource** object.
This method can raise a **DOMASException** object or a **DOMSystemException** object.

Object **DOMASWriter**
DOMASWriter has the all the properties and methods of the **DOMWriter** object as well as the properties and methods defined below.
The **DOMASWriter** object has the following methods:
writeASModel(destination, model)
This method has no return value.
The **destination** parameter is a **DOMOutputStream** object.
The **model** parameter is a **ASModel** object.
This method can raise a **DOMSystemException** object.

Prototype Object **DOMImplementationLS**
The **DOMImplementationLS** class has the following constants:
DOMImplementationLS.MODE_SYNCHRONOUS
This constant is of type **Number** and its value is **1**.
DOMImplementationLS.MODE_ASYNCHRONOUS
This constant is of type **Number** and its value is **2**.

Object **DOMImplementationLS**

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

createDOMBuilder(mode)

This method returns a **DOMBuilder** object.
The **mode** parameter is of type **Number**.
This method can raise a **DOMException** object.

createDOMWriter()

This method returns a **DOMWriter** object.

createDOMInputSource()

This method returns a **DOMInputSource** object.

Prototype Object **DOMBuilder**

The **DOMBuilder** class has the following constants:

DOMBuilder.ACTION_REPLACE

This constant is of type **Number** and its value is **1**.

DOMBuilder.ACTION_APPEND

This constant is of type **Number** and its value is **2**.

DOMBuilder.ACTION_INSERT_AFTER

This constant is of type **Number** and its value is **3**.

DOMBuilder.ACTION_INSERT_BEFORE

This constant is of type **Number** and its value is **4**.

Object **DOMBuilder**

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

entityResolver

This property is a **DOMEntityResolver** object.

errorHandler

This property is a **DOMErrorHandler** object.

filter

This property is a **DOMBuilderFilter** object.

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

setFeature(name, state)

This method has no return value.
The **name** parameter is of type **String**.
The **state** parameter is of type **Boolean**.
This method can raise a **DOMException** object.

canSetFeature(name, state)

This method returns a **Boolean**.
The **name** parameter is of type **String**.
The **state** parameter is of type **Boolean**.

getFeature(name)

This method returns a **Boolean**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

parseURI(uri)

This method returns a **Document** object.
The **uri** parameter is of type **String**.
This method can raise a **DOMSystemException** object.

parse(is)
This method returns a **Document** object.
The **is** parameter is a **DOMInputSource** object.
This method can raise a **DOMSystemException** object.

parseWithContext(is, cnode, action)
This method has no return value.
The **is** parameter is a **DOMInputSource** object.
The **cnode** parameter is a **Node** object.
The **action** parameter is of type **Number**.
This method can raise a **DOMException** object.

Object **DOMWriter**

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

encoding
This property is of type **String**.

lastEncoding
This read-only property is of type **String**.

newLine
This property is of type **String**.

errorHandler
This property is a **DOMErrorHandler** object.

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

setFeature(name, state)
This method has no return value.
The **name** parameter is of type **String**.
The **state** parameter is of type **Boolean**.
This method can raise a **DOMException** object.

canSetFeature(name, state)
This method returns a **Boolean**.
The **name** parameter is of type **String**.
The **state** parameter is of type **Boolean**.

getFeature(name)
This method returns a **Boolean**.
The **name** parameter is of type **String**.
This method can raise a **DOMException** object.

writeNode(destination, wnode)
This method returns a **Boolean**.
The **destination** parameter is a **DOMOutputStream** object.
The **wnode** parameter is a **Node** object.
This method can raise a **DOMSystemException** object.

writeToString(wnode)
This method returns a **String**.
The **wnode** parameter is a **Node** object.
This method can raise a **DOMException** object.

Object **DOMInputSource**

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

byteStream
This property is a **DOMInputSource** object.
characterStream
This property is a **DOMReader** object.
stringData
This property is of type **String**.
encoding
This property is of type **String**.
publicId
This property is of type **String**.
systemId
This property is of type **String**.
baseURI
This property is of type **String**.

Object **LSLoadEvent**
LSLoadEvent has the all the properties and methods of the **Event** object as well as the properties and methods defined below.
The **LSLoadEvent** object has the following properties:
newDocument
This read-only property is a **Document** object.
inputSource
This read-only property is a **DOMInputSource** object.

Object **LSProgressEvent**
LSProgressEvent has the all the properties and methods of the **Event** object as well as the properties and methods defined below.
The **LSProgressEvent** object has the following properties:
inputSource
This read-only property is a **DOMInputSource** object.
position
This read-only property is of type **Number**.
totalSize
This read-only property is of type **Number**.

Object **DOMEntityResolver**
The **DOMEntityResolver** object has the following methods:
resolveEntity(publicId, systemId, baseURI)
This method returns a **DOMInputSource** object.
The **publicId** parameter is of type **String**.
The **systemId** parameter is of type **String**.
The **baseURI** parameter is of type **String**.
This method can raise a **DOMSystemException** object.

Object **DOMBuilderFilter**
The **DOMBuilderFilter** object has the following properties:
whatToShow
This read-only property is of type **Number**.
The **DOMBuilderFilter** object has the following methods:

startNode(snode)

This method returns a **Number**.

The **snode** parameter is a **Node** object.

endNode(enode)

This method returns a **Number**.

The **enode** parameter is a **Node** object.

Object **DOMWriterFilter**

DOMWriterFilter has the all the properties and methods of the **NodeFilter** object as well as the properties and methods defined below.

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

whatToShow

This read-only property is of type **Number**.

Object **DocumentLS**

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

async

This property is of type **Boolean**.

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

abort()

This method has no return value.

load(uri)

This method returns a **Boolean**.

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

loadXML(source)

This method returns a **Boolean**.

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

saveXML(snode)

This method returns a **String**.

The **snode** parameter is a **Node** object.

This method can raise a **DOMException** object.

Object **ParseErrorEvent**

ParseErrorEvent has the all the properties and methods of the **Event** object as well as the properties and methods defined below.

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

error

This read-only property is a **DOMError** object.

Appendix D: Acknowledgements

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

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

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

D.1: Production Systems

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

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

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

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

Glossary

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

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

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

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

child
A *child* is an immediate descendant node of a node.

content model
The *content model* is a simple grammar governing the allowed types of the child elements and the order in which they appear. See *Element Content* in XML [XML].

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

element
Each document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a value. See *Logical Structures* in XML [XML].

live
An object is *live* if any change to the underlying document structure is reflected in the object.

local name
A *local name* is the local part of a *qualified name*. This is called the local part in Namespaces in XML [XML Namespaces].

namespace prefix
A *namespace prefix* is a string that associates an element or attribute name with a *namespace URI* in XML. See namespace prefix in Namespaces in XML [XML Namespaces].

namespace URI
A *namespace URI* is a URI that identifies an XML namespace. This is called the namespace name in Namespaces in XML [XML Namespaces].

partially valid
A node in a DOM tree is *partially valid* if it is *well formed* [p.136] (this part is for comments and processing instructions) and its immediate children are those expected by the content model. The node may be missing trailing required children yet still be considered *partially valid*.

qualified name

A *qualified name* is the name of an element or attribute defined as the concatenation of a *local name* (as defined in this specification), optionally preceded by a *namespace prefix* and colon character. See *Qualified Names* in Namespaces in XML [XML Namespaces].

tokenized

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

well-formed document

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

XML

Extensible Markup Language (*XML*) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. [XML]

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
W3C (World Wide Web Consortium) Character Model for the World Wide Web, January 2001. Available at http://www.w3.org/TR/2001/WD-charmod-20010126

DOM Level 3 Core
W3C (World Wide Web Consortium) Document Object Model Level 3 Core Specification, September 2001. Available at http://www.w3.org/TR/DOM-Level-3-Core

ECMAScript
ISO (International Organization for Standardization). ISO/IEC 16262:1998. ECMAScript Language Specification. Available from ECMA (European Computer Manufacturers Association) at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

ISO/IEC 10646
ISO (International Organization for Standardization). ISO/IEC 10646-1:2000 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization.

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

OMGIDL
OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from http://www.omg.org

RFC2396
IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt

RFC3023
IETF (Internet Engineering Task Force) RFC 3023: XML Media Types, eds. M. Murata, S. St.Laurent, D. Kohn. Available at http://www.ietf.org/rfc/rfc3023.txt

SAX
Simple API for XML, David Megginson. Available at http://www.megginson.com/SAX

Unicode 3.0
The Unicode Consortium. The Unicode Standard, Version 3.0., 2000, Reading, Mass.: Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5.

XML
W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, October 2000. Available at http://www.w3.org/TR/2000/REC-xml-20001006

XML Information set
W3C (World Wide Web Consortium) XML Information Set, October 2001. Available at http://www.w3.org/TR/2001/REC-xml-infoset-20011024

XML Namespaces
W3C (World Wide Web Consortium) Namespaces in XML, January 1999. Available at http://www.w3.org/TR/1999/REC-xml-names-19990114

XML Schema Part 0
W3C (World Wide Web Consortium) XML Schema Part 0, May 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-0-20010502

XML Schema Part 1
W3C (World Wide Web Consortium) XML Schema 1: Structures, May 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502

XML Schema Part 2
W3C (World Wide Web Consortium) XML Schema 2: Datatypes, May 2001. Available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502

F.2: Informative references

Canonical XML
W3C (World Wide Web Consortium) Canonical XML, March 2001. Available at http://www.w3.org/TR/2001/REC-xml-c14n-20010315

COM
Microsoft Corporation The Component Object Model. Available at http://www.microsoft.com/com

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

RFC2616
IETF (Internet Engineering Task Force) RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1 , eds. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt

Index