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

Version 1.0

W3C Working Draft 09 April 2002

This version:
http://www.w3.org/TR/2002/WD-DOM-Level-3-ASLS-20020409
(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/2002/WD-DOM-Level-3-ASLS-20020114

Editors:
Ben Chang, *Oracle*
Elena Litani, *IBM*
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.

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.

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
Elena Litani, IBM
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) feature. This module provides a representation for *XML abstract schemas*, e.g., DTDs [XML 1.0] 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.164] documents used in both the document editing and abstract schema editing worlds.

A DOM application can use the `hasFeature` method of the `DOMImplementation` interface to determine whether a given DOM supports these capabilities or not. This module defines 3 feature strings: `"AS-READ"` for read only access to abstract schemas, `"AS-EDIT"` editing of abstract schemas, and `"AS-DOC"` for document-editing interfaces.

This chapter interacts strongly with Document Object Model Load and Save [p.69] . 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.

1.2. Issue List

1.2.1. Open Issues

Issue AS-Issue-8:
: For Abstract Schemas interfaces (AS-READ/AS-EDIT) there should be no dependency on DOM Core. How can we remove inheritance between `DOMImplementationAS` [p.52] and DOMImplementation?

Issue AS-Issue-9:
: ASDatatype. Can we remove OTHER_SIMPLE_DATATYPE, COMPLEX_DATATYPE?

Issue AS-Issue-10:
: ASDatatype. Do we need to add ANY_TYPE, ANY_SIMPLETYPE?

Issue AS-Issue-11:
: ASWObjectList is live. What does it mean for ASObjectList to be live?

Issue AS-Issue-12:
: Should we rename `ASAttributeDecl.defaultType` [p.30] attribute to `constraintType`?

Issue AS-Issue-13:
: Should default value constraint be added to ASDataType? This would include defaultType constraint and defaultValue.

Issue AS-Issue-14:
: Clarify what exactly `ASWModel` [p.32] represents: does it represent "a schema" as defined in XML Schema or a schema document? If it represents a "document" we should consider removing inheritance between ASModel and ASWModel interfaces (the inheritance between components should be kept).

Issue AS-Issue-15:
: Is there any need for `ASWModel.insertASObject` [p.36] method? Can we remove this method?

Issue AS-Issue-16:
: Since each object carries a namespace do we need setNamedItemNS and removeNamedItemNS methods on `ASWObjectList`?

Issue AS-Issue-17:
: Is there need for separation between `ASObject.rawname` [p.19] and `ASObject.name` [p.19] ? Can "name" be defined as one attribute? If we need to keep `ASObject.rawname`, can we come up with another name? Qualified name can not be used because it does not include multiple colons. Also, the factory method have parameter name, should those include rawname as well?

Issue AS-Issue-18:
: `ASDOMBuilder` [p.65] allows to parse a schema document. The return type is ASWModel. This means that implementation that want to implement Loading of schemas must implement AS-EDIT. The return type should be a read-only object.

Issue AS-Issue-19:
: `ASDOMBuilder.abstractSchema` [p.66] allows to set a schema on a parser instance. How can user set multiple schemas?

Issue AS-Issue-20:
: Does ASDOMWriter 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?

Issue AS-Issue-21:
: The constants have a common suffix. Would not it easier to find the constants within same group alphabetically? In other words, should constants include common prefix instead?

Issue AS-Issue-22:
: `DOMImplementationAS.createASModel` takes as a parameter schemaType. What happens if user specifies `null` for the schemaType?

1.2.2. Resolved Issues

Issue AS-Issue-1:
: Should we add a constant for a datatype? Should ASDataType inherit from ASObject?
: **Resolution:** ASObject does not need to model datatypes.

Issue AS-Issue-2:
: INTERNAL_SUBSET, EXTERNAL_SUBSET, INTERNAL_ENTITY and EXTERNAL_ENTITY seem to carry the same meaning. Should _ENTITY constants be removed?
: **Resolution:** No. The constants are used for different purposes and on different interfaces. It would be confusing for users to have same constant for 2 different purposes.

Issue AS-Issue-3:
: Some components in Abstract Schema do not have name. ASObject serves as a base component for the model. Should it include names/namespace fields?
: **Resolution:** Yes. It gives more flexibility to a user while manipulating/accessing objects in a list: no

cast to type-specific interface is needed.

Issue AS-Issue-4:
: We should remove ASElementDecl.isPCDataOnly and corresponding setPCDataOnly methods. Instead, lets introduce SIMPLE content type that will represent PCDataOnly for DTDs. In general, simple content means that conent consists only of character data (there are no elements in the content).

  Resolution: Remove isPCDataOnly methods. Add SIMPLE_CONTENTTYPE constant.

Issue AS-Issue-5:
: ASAttributeDecl.enumData is DTD only field. Can we remove this field?

  Resolution: No.

Issue AS-Issue-6:
: ASAttributeDecl.ownerElements became a readonly field. Implementation are expected to compute this field.

  Resolution: Yes, this should be a readonly field.

Issue AS-Issue-7:
: For the RW AS editing interfaces, exceptions need to be thrown for such operations if the AS-READ feature string is set.

  Resolution: No. The implementation will always support AS-EDIT in this mode, no changing back and forth

1.2.2.1. General Issues

Issue 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 `NamespaceAware` attribute to the generic AS object to help applications determine which of these fields are important, but we are still analyzing this challenge. Nonetheless, after much discussion, we have made the decision that only 1 active ASModel is allowed, even on ASDOMBuilder (the parser would set the 1 active schema).

Issue I2:
: 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.

Issue I3:
: 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.

  Resolution: Won't deal w/this

Issue I4:
: 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.

Issue I5:

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.

Issue 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.

Resolution: Won’t deal w/this for now.

Issue I7:

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.

Issue I8:

Rename the document-editing interfaces so they should have uniform names such as NodeEditAS, DocumentEditAS, ElementEditAS, etc.

Issue I9:

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.

Issue I10:

If another `ASModel` [p.25] 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.

Issue I11:

List of ASExceptions in the AS spec thus far: INVALID_CHARACTER_ERR, DUPLICATE_NAME_ERR, VALIDATION_ERR.

Issue 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. Will be done in CORE.

Issue 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. It was decided that the validate method be on DocumentAS.

Issue I14:

If entities are changed in the ASWModel, the underlying model is unchanged until normalization.

Issue I15:

Add option to control whether DOM AS is built from this document - solution is that the model is loaded (if there is one) and can be retrieved through the DocumentAS interface.

Issue I16:

There is a way to add a new schema file to the existing active compound schema via addASWModel().

Issue I17:

Altering the document during error reporting, or mutation during validation terminates validation, and a warning will be produced if this happens.

Issue I18:
: Proposal needed to rename the asHint, asLocation attributes and tie that into how to describe an `ASWModel` [p.32] container of other ASWModels.

Issue I19:
: Proposal to revise getElementDecl method and introduce other methods on the DocumentAS interface, such as getAttributeDecl, getNotationDecl, getEntityDecl. Some have mentioned that that these methods should better be added on ASWModel to distinguish between where these declarations came from, the internal or external subset. RESOLUTION: added get*Decl methods on DocumentAS.

Issue I20:
: If implementation doesn’t support AS-editing, need to have each set method throw an unsupported exception. DONE.

Issue I21:
: Notion of read-only AS to be discussed. Currently, the activeASModel is read/write. Need to see where appropriate exceptions need to be thrown in a read-only AS. The methods affected are the following: addASWModel, removeAS, importASObject, insertASObject, removedNamedItem, setNamedItem, addASWAttributeDecl, removeASWAttributeDecl, removesubModel, insertBeforesubModel, appendsubModel, setInternalAS, addAS, removeAS. For a read-only AS, the NO_MODIFICATION exception would be thrown when the feature string AS-READ is set. **Resolution:** The interfaces for Abstract Schema were split into read and read/write. New feature string introduced "AS-READ".

Issue I22:
: For developers, need to keep ASWElementDecl in sync for both an `ASWModel` [p.32] and ASWContentModel, if operations modify the ASWElementDecl. Documentation was added in the ASWElementDecl editing methods to reflect this.

Issue I23:
: Need to clarify the descriptions on the `ASWModel` [p.32] internal/external subset and "global" attributes. Need to also introduce the concept of a dummy element declaration and dummy constant for element types not yet declared but appearing in the content model of another element type.

Issue I24:
: Is there a need for ElementEditAS.getASWElementDecl()? No longer in existence.

Issue I25:
: What happens when a user adds DOCTYPE nodes, when do you create a DOCTYPE after a change in ASWModel or after the document has been created? The "can be resolved" is done via the "normalizeDocument" method since it will be able to update the DOCTYPE node according to the abstractschema.

Issue I26:
: TO BE DONE: Need to add a table for ASObject that is analogous to the table in DOM CORE for Node.

Issue I27:
: TO BE DONE: "Unknown" validity needs to be accounted for validity states.

Issue I28:
: Why have both setNamedItemNS and setNamedItem; and removeNameItemNS and removeNamedItem? One set can be eliminated, but CORE has similar functionality.

1.3. Abstract Schema API

This section defines APIs for accessing, manipulating and modifying an abstract schema (database schema, DTD, XML Schema).

1.3.1. Basic Abstract Schema Interfaces

The set of Abstract Schema interfaces that are common for AS-READ and AS-EDIT.

Interface *ASConstants*

This interface defines a set of constants used in AS model.

IDL Definition

```
interface ASConstants {

  // ASObject Types
  const unsigned short      ELEMENT_DECLARATION            = 1;
  const unsigned short      ATTRIBUTE_DECLARATION          = 2;
  const unsigned short      NOTATION_DECLARATION           = 3;
  const unsigned short      ENTITY_DECLARATION             = 4;
  const unsigned short      CONTENT_MODEL                  = 5;
  const unsigned short      SCHEMA_MODEL                   = 6;


  // Schema Model types
  const unsigned short      INTERNAL_SUBSET                = 30;
  const unsigned short      EXTERNAL_SUBSET                = 31;
  const unsigned short      NOT_USED                       = 32;


  // Entity Types
  const unsigned short      INTERNAL_ENTITY                = 33;
  const unsigned short      EXTERNAL_ENTITY                = 34;


  // Content Model Types
  const unsigned short      EMPTY_CONTENTTYPE              = 40;
  const unsigned short      SIMPLE_CONTENTTYPE             = 41;
  const unsigned short      ELEMENT_CONTENTTYPE            = 42;
  const unsigned short      MIXED_CONTENTTYPE              = 43;
  const unsigned short      ANY_CONTENTTYPE                = 44;


  // Content model compositors
  const unsigned short      SEQUENCE_CM                    = 50;
  const unsigned short      CHOICE_CM                      = 51;
  const unsigned short      ALL_CM                         = 52;
  const unsigned short      UNDEFINED_CM                   = 53;
  const unsigned short      ATOMIC_CM                      = 54;


  // Value Constraint
```

```
    const unsigned short      NONE_VC                           = 0;
    const unsigned short      DEFAULT_VC                        = 60;
    const unsigned short      FIXED_VC                          = 61;
    const unsigned short      REQUIRED_VC                       = 62;


    // Definition of unbounded
    const unsigned long       UNBOUNDED                         = MAX_VALUE;
  };
```

Definition group *ASObject Types*

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

Defined Constants

`ATTRIBUTE_DECLARATION`
: The object describes an attribute declaration.

`CONTENT_MODEL`
: The object describes a content model definition.

`ELEMENT_DECLARATION`
: The object describes an element declaration.

`ENTITY_DECLARATION`
: The object describes an entity declaration.

`NOTATION_DECLARATION`
: The object describes a notation declaration.

`SCHEMA_MODEL`
: The object describes a schema model.

Definition group *Schema Model types*

A code representing how the `ASModel` [p.25] is used.

Defined Constants

`EXTERNAL_SUBSET`
: The `ASModel` [p.25] is used as an external subset.

`INTERNAL_SUBSET`
: The `ASModel` [p.25] is used as an internal subset.

`NOT_USED`
: The `ASModel` [p.25] is neither used as an internal or external subset.

Definition group *Entity Types*

An integer indicating which type of entity this is.

Defined Constants

`EXTERNAL_ENTITY`
: A constant defining an external entity.

`INTERNAL_ENTITY`
: A constant defining an internal entity.

Definition group *Content Model Types*

A code representing the type of content model.

Defined Constants

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

`ELEMENT_CONTENTTYPE`
: Represents an element-only content type. An element-only content type validates elements with children that conform to the supplied content model.

`EMPTY_CONTENTTYPE`
: Represents an EMPTY content type for an Element declaration. A content type with the distinguished value empty validates elements with no character or element information item children.

`MIXED_CONTENTTYPE`
: Represents a MIXED content type for an Element declaration.

`SIMPLE_CONTENTTYPE`
: The content model type simple. A content type which is a simple validates elements with character-only children.

Definition group *Content model compositors*

List of content model compositors.

Defined Constants

`ALL_CM`
: This content model represents a simplified version of the SGML &-Connector and is limited to the top-level of any content model. No element in the content model may appear more than once. Please refer to the definition element-all.

`ATOMIC_CM`
: In this content model the `subModel` includes a single definition.

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

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

`UNDEFINED_CM`
: This content model is undefined, and is associated with incomplete element declarations in the ASModel, meaning elements implicitly declared through an attribute list but without any corresponding element declarations.

Definition group *Value Constraint*

Defined Constants

`DEFAULT_VC`
: Indicates that there is a default value constraint.

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

`NONE_VC`
: Describes that the component does not have any value constraint.

`REQUIRED_VC`
Indicates that attribute is required.

Definition group *Definition of unbounded*

Defined Constants

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

Interface *ASObject*

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

IDL Definition

```
interface ASObject {
  readonly attribute unsigned short  objectType;
  readonly attribute ASModel         ownerModel;
  readonly attribute DOMString       rawname;
  readonly attribute DOMString       name;
  readonly attribute DOMString       namespace;
  ASObject           cloneASObject(in boolean deep)
                                        raises(ASException);
};
```

Attributes

`name` of type `DOMString`, readonly
The name of type `NCName` of this declaration as defined in [XML Namespaces].

`namespace` of type `DOMString`, readonly
The *namespace URI* [p.164] of this object, or `null` if it is unspecified. [XML Schema Part 1]defines how a *namespace URI* [p.164] is attached to schema components.

`objectType` of type `unsigned short`, readonly
The `type` of this object, ei. `ELEMENT_DECLARATION`.

`ownerModel` of type `ASModel` [p.25] , readonly
The `ASModel` [p.25] object associated with this `ASObject`. For a object of type `AS_MODEL`, this is `null`.

`rawname` of type `DOMString`, readonly
The `rawname` of this declaration of type `Name` as defined in [XML 1.0]. This value will be available only for schemas that allow to declare name of type `Name`.

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`.

Exceptions

`ASException` [p.48]	NOT_SUPPORTED_ERR: Raised if implementation doesn't support AS-EDIT.

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      ANYSIMPLETYPE_DATATYPE         = 216;
  const unsigned short      ANYTYPE_DATATYPE               = 216;
};
```

Definition group *DATA_TYPES*

An integer indicating which datatype this is.

Defined Constants

`ANYSIMPLETYPE_DATATYPE`
: A code representing a *anySimpleType* data type as defined in [XML Schema Part 2].

`ANYTYPE_DATATYPE`
: A code representing a *anyType* data type as defined in [XML Schema Part 2].

`ANYURI_DATATYPE`
: A code representing an *uri reference* data type as defined in [XML Schema Part 2].

`BASE64BINARY_DATATYPE`
: A code representing a *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`
: A code representing a *byte* data type as defined in [XML Schema Part 2].

`DATETIME_DATATYPE`
: A code representing a *datetime* data type as defined in [XML Schema Part 2].

`DATE_DATATYPE`
: A code representing a *date* data type as defined in [XML Schema Part 2].

`DECIMAL_DATATYPE`
: A code representing a *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`
: A code representing a *duration* data type as defined in [XML Schema Part 2].

`ENTITIES_DATATYPE`
: A code representing a *ENTITIES* data type as defined in [XML Schema Part 2].

`ENTITY_DATATYPE`
: A code representing a *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`
: A code representing a *day* data type as defined in [XML Schema Part 2].

`GMONTHDAY_DATATYPE`
: A code representing a *monthday* data type as defined in [XML Schema Part 2].

`GMONTH_DATATYPE`
: A code representing a *month* data type as defined in [XML Schema Part 2].

`GYEARMONTH_DATATYPE`
: A code representing a *yearmonth* data type as defined in [XML Schema Part 2].

`GYEAR_DATATYPE`
: A code representing a *year* data type as defined in [XML Schema Part 2].

`HEXBINARY_DATATYPE`
: A code representing a *hexbinary* data type as defined in [XML Schema Part 2].

`IDREFS_DATATYPE`
: A code representing a *IDREFS* data type as defined in [XML Schema Part 2].

`IDREF_DATATYPE`
: A code representing a *IDREF* data type as defined in [XML Schema Part 2].

`ID_DATATYPE`
: A code representing a *ID* data type as defined in [XML Schema Part 2].

`INTEGER`
: A code representing a *integer* data type as defined in [XML Schema Part 2].

`INT_DATATYPE`
: A code representing a *integer* data type as defined in [XML Schema Part 2].

`LANGUAGE_DATATYPE`
: A code representing a *Language* data type as defined in [XML Schema Part 2].

`LONG_DATATYPE`
: A code representing an *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`
: A code representing an *negative integer* data type as defined in [XML Schema Part 2].

`NMTOKENS_DATATYPE`
: A code representing a *NMTOKENS* data type as defined in [XML Schema Part 2].

`NMTOKEN_DATATYPE`
: A code representing a *NMTOKEN* data type as defined in [XML Schema Part 2].

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

`NONPOSITIVEINTEGER_DATATYPE`
: A code representing a *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`
: A code representing a *NOTATION* data type as defined in [XML Schema Part 2].

`POSITIVEINTEGER_DATATYPE`
: A code representing a *positive integer* data type as defined in [XML Schema Part 2].

`QNAME_DATATYPE`
: A code representing an *XML qualified name* data type as defined in [XML Schema Part 2].

`SHORT_DATATYPE`
: A code representing a *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`
: A code representing a *time* data type as defined in [XML Schema Part 2].

`TOKEN_DATATYPE`
: A code representing a *token* data type as defined in [XML Schema Part 2].

`UNSIGNEDBYTE_DATATYPE`
: A code representing a *unsigned byte* data type as defined in [XML Schema Part 2].

`UNSIGNEDINT_DATATYPE`
: A code representing a *unsigned integer* data type as defined in [XML Schema Part 2].

`UNSIGNEDLONG_DATATYPE`
: A code representing a *unsigned long* data type as defined in [XML Schema Part 2].

`UNSIGNEDSHORT_DATATYPE`
: A code representing a *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 *ASObjectList*

The `ASObjectList` interface provides the abstraction of an ordered collection of AS objects, without defining or constraining how this collection is implemented. `ASObjectList` objects in the DOM AS are *live* [p.163] .

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.163] object 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 objects 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 objects that can be accessed by name. Note that `ASNamedObjectMap` does not inherit from `ASObjectList` [p.23] ; `ASNamedObjectMaps` are not maintained in any particular order.

IDL Definition

```
interface ASNamedObjectMap {
  readonly attribute unsigned long   length;
  ASObject           item(in unsigned long index);
  ASObject           getNamedItem(in DOMString name);
  ASObject           getNamedItemNS(in DOMString namespaceURI,
                                    in DOMString localName);
};
```

Attributes

`length` of type `unsigned long`, readonly
The number of `ASObjects` [p.19] in the `ASObjectList` [p.23] . The range of valid *child* [p.163] object indices is 0 to `length-1` inclusive.

Methods

`getNamedItem`
Retrieves an `ASObject` [p.19] specified by name.
Parameters
`name` of type `DOMString`
The `objectName` of an `ASObject` [p.19] to retrieve.
Return Value

`ASObject` [p.19]	An `ASObject` with specified object name and `null` if the map does not contain an *element* [p.163] with the given name.

No Exceptions

`getNamedItemNS`
Retrieves a node specified by local name and namespace URI.
Per [XML Namespaces], applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.
Parameters

`namespaceURI` of type `DOMString`
The *namespace URI* [p.164] of the node to retrieve.
`localName` of type `DOMString`
The *local name* [p.163] of the node 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 node in this map.

No Exceptions

`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 objects 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` [p.23] , or `null` if that is not a valid index.

No Exceptions

1.3.2. Read Only Abstract Schemas interfaces

The interfaces in this section provide a read-only access to abstract schemas.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "AS-READ" and "3.0" (respectively) to determine whether or not this module is supported by the implementation.

Interface *ASModel*

A read-only interface that represents an abstract schema.

IDL Definition

```
interface ASModel : ASObject {
  readonly attribute boolean         namespaceAware;
  readonly attribute unsigned short  usage;
  readonly attribute DOMString       location;
  readonly attribute DOMString       hint;
  readonly attribute boolean         container;
  ASNamedObjectMap   getComponents(in unsigned short objectType);

  // Convenience method to retrive named top-level declarations
  ASElementDecl      getElementDecl(in DOMString name, 
                                    in DOMString targetNamespace);
  ASAttributeDecl    getAttributeDecl(in DOMString name, 
                                      in DOMString targetNamespace);
  ASEntityDecl       getEntityDecl(in DOMString name);
  ASNotationDecl     getNotationDecl(in DOMString name, 
                                     in DOMString targetNamespace);
};
```

Definition group *Convenience method to retrive named top-level declarations*

Attributes

`container` of type `boolean`, readonly

If `usage` is EXTERNAL_SUBSET or NOT_USED, then the `ASModel` is simply a container of other ASModels.

`hint` of type `DOMString`, readonly

The hint to locating an ASModel. For example, if an `ASModel` modeled a DTD, this could represent the public identifier; if an `ASModel` modeled a XML schema, this could represent a target namespace of a schema document. This attribute can also be NULL.

`location` of type `DOMString`, readonly

The URI reference. For example, if an `ASModel` modeled a DTD, this could represent the system identifier; if an `ASModel` modeled a XML schema, this could act as a hint to the location of a schema document. In addition, if a system identifier doesn't exist for an internet subset, then this attribute can be NULL.

`namespaceAware` of type `boolean`, readonly

`true` if this `ASModel` defines the document structure is namespace-aware [XML Namespaces]; `false` if the document structure is non-namespace-aware.

`usage` of type `unsigned short`, readonly

Uses INTERNAL_SUBSET, EXTERNAL_SUBSET, or NOT_USED. An exception will be raised if it is incompatibly shared or in use as an internal subset.

Methods

`getAttributeDecl`

Returns a top-level attribute declaration.

Parameters

`name` of type `DOMString`

The name of the declaration.

`targetNamespace` of type `DOMString`

The namespace of the declaration, otherwise null.

Return Value

`ASAttributeDecl` [p.29]	A top-level attribute declaration or null if such declaration does not exist.

No Exceptions

`getComponents`

Returns a list of top-level component declarations: ei. element declarations, attribute declarations, etc.

Parameters

`objectType` of type `unsigned short`
The type of the declaration: ELEMENT_DECLARATION, ATTRIBUTE_DECLARATION, etc.
The parameter value SCHEMA_MODEL will retrieve a list of nested or imported ASModels if such list is available.

Return Value

`ASNamedObjectMap` [p.24]	A list of top-level definition of the specified type in `objectType` or `null`.

No Exceptions

`getElementDecl`
Returns a top-level element declaration.

Parameters

`name` of type `DOMString`
The name of the declaration.

`targetNamespace` of type `DOMString`
The namespace of the declaration, otherwise null.

Return Value

`ASElementDecl` [p.28]	A top-level element declaration or null if such declaration does not exist.

No Exceptions

`getEntityDecl`
Returns an entity declaration.

Parameters

`name` of type `DOMString`
The name of the declaration.

Return Value

`ASEntityDecl` [p.30]	An entity declaration or null if such declaration does not exist.

No Exceptions

`getNotationDecl`
Returns a top-level notation declaration.

Parameters

`name` of type `DOMString`
The name of the declaration.

`targetNamespace` of type `DOMString`
The namespace of the declaration, otherwise null.

Return Value

`ASNotationDecl` [p.30]	A top-level notation declaration or null if such declaration does not exist.

No Exceptions

Interface *ASContentModel*

The content model of a declared element.

IDL Definition

```
interface ASContentModel : ASObject {
  readonly attribute unsigned short  contentModelType;
  readonly attribute unsigned long   minOccurs;
  readonly attribute unsigned long   maxOccurs;
  readonly attribute ASObjectList    subModels;
};
```

Attributes

`contentModelType` of type `unsigned short`, readonly
: One of `CHOICE_CM`, `SEQUENCE_CM`, `ALL_CM` or `ATOMIC_CM`. The operator is applied to all the components(ASObjects) in the the `subModels`. For example, if the list operator is `CHOICE_CM` 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`, readonly
: 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`, readonly
: min occurrence for this content particle. Its value may be 0 or a positive integer.

`subModels` of type `ASObjectList` [p.23] , readonly
: Pointers to `ASObject` [p.19] s of the following types: `ELEMENT_DECLARATION` and `CONTENT_MODEL`.

Interface *ASElementDecl*

This interface represents an element declararation.

IDL Definition

```
interface ASElementDecl : ASObject {
  readonly attribute ASDataType      type;
  readonly attribute unsigned short  contentType;
  readonly attribute boolean         strictMixedContent;
  readonly attribute ASContentModel  contentModel;
  readonly attribute boolean         isPCDataOnly;
  readonly attribute ASNamedObjectMap attributeDecls;
  ASAttributeDecl    getAttributeDecl(in DOMString name,
                                       in DOMString targetNamespace);
};
```

Attributes

`attributeDecls` of type `ASNamedObjectMap` [p.24] , readonly
: The`ASNamedObjectMap` [p.24] containing `ASAttributeDecls` [p.29] for all the attributes that can appear on this type of element.

`contentModel` of type `ASContentModel` [p.28] , readonly
: The *content model* [p.163] of element.

`contentType` of type `unsigned short`, readonly
: The content type of the element. One of `EMPTY_CONTENTTYPE`, `SIMPLE_CONTENTTYPE`, `ELEMENT_CONTENTTYPE`, `MIXED_CONTENTTYPE`,`ANY_CONTENTTYPE`.

`isPCDataOnly` of type `boolean`, readonly
: 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`, readonly
: A boolean defining whether the element order and number of the *child* [p.163] 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.163] elements are not important.

`type` of type `ASDataType` [p.20] , readonly
: Datatype of the element.

Methods

`getAttributeDecl`
: A convenience method to get an attribute declaration by name.

Parameters

`name` of type `DOMString`
: The name of the declaration.

`targetNamespace` of type `DOMString`
: The namespace of the declaration, otherwise null.

Return Value

`ASAttributeDecl` [p.29]	A top-level attribute declaration or null if such declaration does not exist.

No Exceptions

Interface *ASAttributeDecl*

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

IDL Definition

```
interface ASAttributeDecl : ASObject {
  readonly attribute ASDataType      type;
  readonly attribute DOMString       enumAttr;
  readonly attribute ASObjectList    ownerElementDeclarations;
  readonly attribute unsigned short  defaultType;
  readonly attribute DOMString       value;
};
```

Attributes

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

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

`ownerElementDeclarations` of type `ASObjectList` [p.23] , readonly
: An `ASObjectList` [p.23] of element declarations that use this `ASAttributeDeclaration`.

`type` of type `ASDataType` [p.20] , readonly
: Datatype of the attribute.

`value` of type `DOMString`, readonly
: Default or fixed value or `null` if there is none.

Interface *ASEntityDecl*

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 in built.)

IDL Definition

```
interface ASEntityDecl : ASObject {
  readonly attribute unsigned short  entityType;
  readonly attribute DOMString       entityValue;
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       publicId;
};
```

Attributes

`entityType` of type `unsigned short`, readonly
: One of the `INTERNAL_ENTITY` or `EXTERNAL_ENTITY`.

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

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

Interface *ASNotationDecl*

This interface represents a notation declaration.

IDL Definition

```
interface ASNotationDecl : ASObject {
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       publicId;
};
```

Attributes

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

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

1.3.3. Abstract Schema Editing Interfaces

A list of the proposed Abstract Schema data structures and functions follow, starting off with the data structures and abstract schema editing methods.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "AS-EDIT" and "3.0" (respectively) to determine whether or not this module is supported by the implementation.

Note that operations on the `ASWModel` [p.32] 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:

```
ASWElementDecl example = {
strictMixedContent    = false;
elementType           = STRING_DATATYPE;
isPCDataOnly          = false;
contentType           = ELEMENTS_CONTENTTYPE;
ASWContentModel        = exE;
ASWAttributeDecls      = null;
}

ASWContentModel exE = {
contentModelType      = SEQUENCE_CM;
minOccurs             = 1;
maxOccurs             = 1;
subModels             = {(ASWElementDecl A),
(ASWContentModel exBC),
(ASWContentModel exD)};
}

ASWElementDecl A = {
strictMixedContent    = false;
elementType           = STRING_DATATYPE;
isPCDataOnly          = false;
contentType           = ELEMENTS_CONTENTTYPE;
ASWContentModel        = null;
ASWAttributeDecls      = null;
}

ASWContentModel exBC = {
contentModelType      = CHOICE_CM;
minOccurs             = 1;
maxOccurs             = 1;
subModels             = {(ASWContentModel exB),
(ASWElementDecl C)};
}

ASWContentModel exB = {
contentModelType      = ATOMIC_CM;
minOccurs             = 0;
maxOccurs             = AS_UNBOUNDED;
subModels             = {(ASWElementDecl B)};
}
ASWElementDecl B = {
strictMixedContent    = false;
elementType           = STRING_DATATYPE;
isPCDataOnly          = false;
contentType           = ELEMENTS_CONTENTTYPE;
ASWContentModel        = null;
ASWAttributeDecls      = null;
}

ASWElementDecl C = {
strictMixedContent    = false;
elementType           = STRING_DATATYPE;
isPCDataOnly          = false;
contentType           = ELEMENTS_CONTENTTYPE;
ASWContentModel        = null;
ASWAttributeDecls      = null;
}

ASWContentModel exD = {
contentModelType     = ATOMIC_CM;
minOccurs            = 1;
maxOccurs            = AS_UNBOUNDED;
subModels            = {(ASWElementDecl D)};
}
ASWElementDecl D = {
strictMixedContent   = false;
elementType          = STRING_DATATYPE;
isPCDataOnly         = false;
contentType          = ELEMENTS_CONTENTTYPE;
ASWContentModel       = null;
ASWAttributeDecls     = null;
}
```

Interface *ASWModel*

To begin with, an abstract schema is a generic structure that could contain both internal and external subsets. An `ASWModel` is an abstract object that could map to a DTD [XML 1.0], an XML Schema [XML Schema Part 0], a database schema, etc. An `ASWModel` could represent either an internal or an external subset; hence an abstract schema could be composed of an `ASWModel` representing the internal subset and an `ASWModel` representing the external subset. Note that the `ASWModel` representing the external subset could consult the `ASWModel` representing the internal subset. Furthermore, the `ASWModel` representing the internal subset could be set to `null` by the setInternalAS() method as a mechanism for "removal". In addition, only one `ASWModel` representing the external subset can be specified as "active" and it is possible that none are "active". Finally, the `ASWModel` contains the factory methods needed to create a various types of ASObjects like `ASWElementDecl` [p.40] , `ASWAttributeDecl` [p.44] , etc.

IDL Definition

```
interface ASWModel : ASModel {
  void               setLocation(in DOMString location);
  void               setHint(in DOMString hint);
  void               addComponent(in ASObject declaration);
  void               removeComponent(in ASObject declaration);
  void               addASModel(in ASModel declaration);
  void               removeASModel(in ASModel declaration);
  ASObjectList       getASModels();
  ASObject           importASObject(in ASObject asobject);
  void               insertASObject(in ASObject asobject);
  boolean            validate();
  ASWElementDecl     createASWElementDecl(in DOMString namespaceURI, 
                                          in DOMString name)
                                        raises(ASException);
  ASWAttributeDecl   createASWAttributeDecl(in DOMString namespaceURI, 
                                            in DOMString name)
                                        raises(ASException);
  ASWNotationDecl    createASWNotationDecl(in DOMString namespaceURI, 
                                           in DOMString name, 
                                           in DOMString systemId, 
                                           in DOMString publicId)
                                        raises(ASException);
  ASWEntityDecl      createASWEntityDecl(in DOMString name)
                                        raises(ASException);
  ASWContentModel    createASWContentModel(in DOMString name, 
                                           in DOMString namespaceURI, 
                                           in unsigned long minOccurs, 
                                           in unsigned long maxOccurs, 
                                           in unsigned short operator)
                                        raises(ASException);
};
```

Methods

`addASModel`

Adds ASModel to the list of ASModels.

Parameters

`declaration` of type `ASModel` [p.25]

An ASModel to be added

No Return Value

No Exceptions

`addComponent`

Add top-level component declaration to a list of those definitions.

Parameters

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

A component declaration.

No Return Value

No Exceptions

`createASWAttributeDecl`

Creates an attribute declaration.

Parameters

`namespaceURI` of type `DOMString`

The *namespace URI* [p.164] 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 ASWModel-dependent.

Return Value

`ASWAttributeDecl` [p.44]	A new `ASWAttributeDecl` object with appropriate attributes set by input parameters.

Exceptions

`ASException` [p.48]	INVALID_CHARACTER_ERR: Raised if the input `name` parameter contains an illegal character.

`createASWContentModel`

Creates an object which describes part of an `ASWElementDecl` [p.40] 's content model.

Parameters

`name` of type `DOMString`

The name of this `ASWContentModel` [p.42] .

`namespaceURI` of type `DOMString`

The namespace URI of this `ASWContentModel` [p.42] .

`minOccurs` of type `unsigned long`

The minimum occurrence for the subModels of this `ASWContentModel` [p.42] .

`maxOccurs` of type `unsigned long`

The maximum occurrence for the subModels of this `ASWContentModel` [p.42] .

`operator` of type `unsigned short`

operator of type `CHOICE_CM`, `SEQUENCE_CM`, `ALL_CM` or `ATOMIC_CM`.

Return Value

`ASWContentModel` [p.42]	A new `ASWContentModel` object.

Exceptions

`ASException` [p.48]	A `ASException` [p.48] , e.g., `minOccurs > maxOccurs`.

`createASWElementDecl`

Creates an *element* [p.163] 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 ASWModel-dependent.
Return Value

`ASWElementDecl` [p.40]	A new `ASWElementDecl` object with the `ASObject` [p.19] `objectName` attribute set to `name` and `namespaceURI` set to `namespaceURI`. Other attributes of the element declaration are set through `ASWElementDecl` interface methods. Depending on the value of `NamespaceAware`, this method will take into account the `namespaceURI` parameter.

Exceptions

`ASException` [p.48]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createASWEntityDecl`
Creates an ASWEntityDecl.
Parameters
`name` of type `DOMString`
The name (of type `Name`) of the entity being declared.
Return Value

`ASWEntityDecl` [p.46]	A new `ASWEntityDecl` object with `entityName` attribute set to name.

Exceptions

`ASException` [p.48]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createASWNotationDecl`
Creates a new notation declaration.
Parameters
`namespaceURI` of type `DOMString`
The *namespace URI* [p.164] 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 ASWModel-dependent.

`systemId` of type `DOMString`
The URI reference for the notation declaration.
`publicId` of type `DOMString`
The public identifier for the notation declaration.

Return Value

`ASWNotationDecl` [p.47]	A new `ASWNotationDecl` object with `notationName` attribute set to `name` and `publicId` and `systemId` set to the corresponding fields.

Exceptions

`ASException` [p.48]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`getASModels`
Returns a list of ASModels.

Return Value

`ASObjectList` [p.23]	A list of ASModels.

No Parameters
No Exceptions

`importASObject`
Imports `ASObject` [p.19] into ASWModel.

Parameters

`asobject` of type `ASObject` [p.19]
`ASObject` to be imported.

Return Value

`ASObject` [p.19]	The `ASObject` that is imported.

No Exceptions

`insertASObject`
Inserts `ASObject` [p.19] into ASWModel.

Parameters

`asobject` of type `ASObject` [p.19]
`ASObject` to be inserted.

No Return Value
No Exceptions

`removeASModel`
Removes ASModel from the list of ASModels.

Parameters

`declaration` of type `ASModel` [p.25]
An ASModel to be removed
No Return Value
No Exceptions

`removeComponent`
Removes the specified `ASObject` [p.19] from the list of top-level declarations.
Parameters
`declaration` of type `ASObject` [p.19]
A component declaration to be removed
No Return Value
No Exceptions

`setHint`
Set hint for the ASWModel.
Parameters
`hint` of type `DOMString`
The hint to locating an ASWModel. For example, if an `ASWModel` modeled a DTD, this could represent the public identifier; if an `ASWModel` modeled a XML schema, this could represent a target namespace of a schema document. This attribute can also be NULL.
No Return Value
No Exceptions

`setLocation`
Set location of schema model.
Parameters
`location` of type `DOMString`
The URI reference. For example, if an `ASWModel` modeled a DTD, this could represent the system identifier; if an `ASWModel` modeled a XML schema, this could act as a hint to the location of a schema document. In addition, if a system identifier doesn’t exist for an internet subset, then this attribute can be NULL.
No Return Value
No Exceptions

`validate`
Determines if an `ASModel` [p.25] 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` [p.25] is valid, `false` otherwise.

No Parameters
No Exceptions

Interface *ASWNamedObjectMap*

Objects implementing the `ASWNamedObjectMap` interface are used to represent collections of abstract schema objects that can be accessed by name. Note that `ASNamedObjectMap` [p.24] does not inherit from `ASObjectList` [p.23] ; `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] .

`ASWNamedObjectMap` object in the DOM are *live* [p.163] .

Issue NamedObjectMap-live:
Should named node map be live?

IDL Definition

```
interface ASWNamedObjectMap : ASNamedObjectMap {
  ASObject           removeNamedItem(in DOMString name)
                                       raises(ASException);
  ASObject           setNamedItem(in ASObject newASObject)
                                       raises(ASException,
                                              ASException);
  ASObject           setNamedItemNS(in ASObject arg)
                                       raises(ASException);
  ASObject           removeNamedItemNS(in DOMString namespaceURI,
                                      in DOMString localName)
                                       raises(DOMException);
};
```

Methods

`removeNamedItem`

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

Parameters

`name` of type `DOMString`

The `objectName` 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

`ASException` [p.48]	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 a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the attributes attribute of the `ASObject` [p.19] interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

Parameters

`namespaceURI` of type `DOMString`
The *namespace URI* [p.164] of the node to remove.
`localName` of type `DOMString`
The *local name* [p.163] of the `ASObject` to remove.
Return Value

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

`setNamedItem`
Adds an `ASObject` [p.19] using its `objectName` 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 `objectName` as the key.
Return Value

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

Exceptions

`ASException` [p.48]	WRONG_MODEL_ERR: Raised if `arg` was created from a different `ASWModel` [p.32] than the one that created this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
`ASException` [p.48]	

`setNamedItemNS`
Adds a node using its `namespaceURI` and `localName`. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one.
Per [XML Namespaces], applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.
Parameters

`arg` of type `ASObject` [p.19]
A node to store in this map. The node will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Return Value

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

Exceptions

`ASException` [p.48]	WRONG_MODEL_ERR: Raised if `arg` was created from a different `ASWModel` [p.32] than the one that created this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

Interface *ASWElementDecl*

The element declaration.

IDL Definition

```
interface ASWElementDecl : ASElementDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespace(in DOMString namespaceURI);
  void               setStrictMixedContent(in boolean mixedContent);
  void               setType(in ASDataType type);
  void               setContentType(in unsigned short contentType);
  void               setContentModel(in ASWContentModel contentModel);
  void               addAttributeDecl(in ASWAttributeDecl attributeDecl);
  ASWAttributeDecl   removeAttributeDecl(in ASWAttributeDecl attributeDecl);
};
```

Methods

`addAttributeDecl`
Adds an `ASWAttributeDecl` [p.44] for the element being declared.
Parameters
`attributeDecl` of type `ASWAttributeDecl` [p.44]
The new attribute declaration to add. If the attribute declaration already exists for the element, the call does not have any effect.
No Return Value
No Exceptions

`removeAttributeDecl`
Removes an `ASWAttributeDecl` [p.44] from the element being declared.
Parameters

`attributeDecl` of type `ASWAttributeDecl` [p.44]
The attribute declaration to be removed. If the attribute declaration does not exist for the element, the call does not have any effect.

Return Value

`ASWAttributeDecl` [p.44]	`null` if the attribute does not exist. Otherwise returns the attribute being removed.

No Exceptions

`setContentModel`
Set the content model for this element declaration.
Parameters
`contentModel` of type `ASWContentModel` [p.42]
The *content model* [p.163] of element.
No Return Value
No Exceptions

`setContentType`
Set content type for this element declaration.
Parameters
`contentType` of type `unsigned short`
The content type of the element. One of `EMPTY_CONTENTTYPE, SIMPLE_CONTENT, ANY_CONTENTTYPE, MIXED_CONTENTTYPE, ELEMENT_CONTENTTYPE`.
No Return Value
No Exceptions

`setName`
Set the name of type `NCName` for this declaration.
Parameters
`name` of type `DOMString`
The name for this declaration.
No Return Value
No Exceptions

`setNamespace`
Set the namespace for this declaration.
Parameters
`namespaceURI` of type `DOMString`
The namespace URI for this declaration.
No Return Value
No Exceptions

`setRawname`
Set the rawname for this declaration.
Parameters
`rawname` of type `DOMString`
The rawname of type `Name` for this declaration.
No Return Value
No Exceptions

`setStrictMixedContent`

Set mixed content.

Parameters

`mixedContent` of type `boolean`

A boolean defining whether the element order and number of the *child* [p.163] 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.163] elements are not important.

No Return Value

No Exceptions

`setType`

Set the type for this element declaration.

Parameters

`type` of type `ASDataType` [p.20]

The datatype for this element declaration.

No Return Value

No Exceptions

Interface *ASWContentModel*

The content model of a declared element.

IDL Definition

```
interface ASWContentModel : ASContentModel {
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setContentModelType(in unsigned short operator);
  void               setMinOccurs(in unsigned long minOccurs);
  void               setMaxOccurs(in unsigned long maxOccurs);
  void               removeSubModel(in ASObject oldObject);
  ASObject           insertBeforeSubModel(in ASObject newObject,
                                          in ASObject refObject)
                                        raises(ASException);
  unsigned long      appendSubModel(in ASObject newObject)
                                        raises(ASException);
};
```

Methods

`appendSubModel`

Appends a new ASObject to the end of the list representing the`subModels`.

Parameters

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

The new object to be appended.

Return Value

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

Exceptions

`ASException` [p.48]	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 `ASWContentModel` nor an `ASWElementDecl` [p.40] .

`insertBeforeSubModel`

Inserts a new object in the submodel before the existing reference object. Objects that already exist in the list are moved as needed.

Parameters

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

The new object to be inserted.

`refObject` of type `ASObject`

The reference object before which the new object is to be inserted.

Return Value

`ASObject` [p.19]	The object being inserted.

Exceptions

`ASException` [p.48]	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 `ASWContentModel` nor an `ASWElementDecl` [p.40] .

`removeSubModel`

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

Parameters

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

The object to be removed.

No Return Value

No Exceptions

`setContentModelType`

Set content model type

Parameters

`operator` of type `unsigned short`

One of `CHOICE_CM`, `SEQUENCE_CM`, `ALL_CM`, `ATOMIC_CM`, or `UNDEFINED_CM`. The operator is applied to all the components(ASObjects) in the the `subModels`. For example, if the content model type is `CHOICE_CM` and the components in subModels are a, b and c then the abstract schema for the element being declared is `(a|b|c)`

No Return Value

No Exceptions

`setMaxOccurs`

Set maxOccurs for the content model

Parameters

`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.

No Return Value

No Exceptions

`setMinOccurs`

Set minOccurs for the content model

Parameters

`minOccurs` of type `unsigned long`

min occurrence for this content particle. Its value may be 0 or a positive integer.

No Return Value

No Exceptions

`setName`

Set the name of type `NCName` for this declaration.

Parameters

`name` of type `DOMString`

The name for this declaration.

No Return Value

No Exceptions

`setNamespaceURI`

Set the namespace URI for this declaration.

Parameters

`namespaceURI` of type `DOMString`

The namespace URI for this declaration.

No Return Value

No Exceptions

Interface *ASWAttributeDecl*

An attribute declaration.

IDL Definition

```
interface ASWAttributeDecl : ASAttributeDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setType(in ASDataType type);
  void               setValue(in DOMString value);
  void               setEnumAttr(in DOMString enumeration);
  void               setDefaultType(in unsigned short constraint);
};
```

Methods

`setDefaultType`

Set constraint for the attribute’s value

Parameters

`constraint` of type `unsigned short`

Constraint type if any for this attribute.

No Return Value

No Exceptions

`setEnumAttr`

Set enumeration value for this attribute

Parameters

`enumeration` of type `DOMString`

Valid attribute values, separated by vertical bars, in a string.

No Return Value

No Exceptions

`setName`

Set the name of type `NCName` for this declaration.

Parameters

`name` of type `DOMString`

The name for this declaration.

No Return Value

No Exceptions

`setNamespaceURI`

Set the namespace URI for this declaration.

Parameters

`namespaceURI` of type `DOMString`

The namespace URI for this declaration.

No Return Value

No Exceptions

`setRawname`

Set the rawname for this declaration.

Parameters

`rawname` of type `DOMString`

The rawname of type `Name` for this declaration.

No Return Value

No Exceptions

`setType`

Set the type for this attribute declaration.

Parameters

`type` of type `ASDataType` [p.20]

The datatype for this attribute declaration.

No Return Value

No Exceptions

`setValue`

Set default or fixed value for this attribute

Parameters

`value` of type `DOMString`

Default or fixed value or `null` if there is none.

No Return Value

No Exceptions

Interface *ASWEntityDecl*

Models a general entity declaration in an abstract schema.

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 ASWEntityDecl : ASEntityDecl {
  void               setRawname(in DOMString rawname);
  void               setEntityType(in unsigned short type);
  void               setEntityValue(in DOMString value);
  void               setSystemId(in DOMString systemId);
  void               setPublicId(in DOMString publicId);
};
```

Methods

`setEntityType`

Set the type for this entity declaration

Parameters

`type` of type `unsigned short`

One of the `INTERNAL_ENTITY` or `EXTERNAL_ENTITY`.

No Return Value

No Exceptions

`setEntityValue`

Set entity value.

Parameters

`value` 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`.

No Return Value

No Exceptions

`setPublicId`

Set publicId for this entity

Parameters

`publicId` of type `DOMString`

The string representing the public identifier for this entity declaration, if present; `null` otherwise.

No Return Value

No Exceptions

`setRawname`

Set the rawname for this declaration.

Parameters

`rawname` of type `DOMString`
The rawname of type `Name` for this declaration.
No Return Value
No Exceptions

`setSystemId`
Set systemId for this entity
Parameters
`systemId` of type `DOMString`
The URI reference representing the system identifier for the entity declaration, if present, `null` otherwise.
No Return Value
No Exceptions

Interface *ASWNotationDecl*

This interface represents a notation declaration.

IDL Definition

```
interface ASWNotationDecl : ASNotationDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setSystemId(in DOMString systemId);
  void               setPublicId(in DOMString publicId);
};
```

Methods

`setName`
Set the name of type `NCName` for this declaration.
Parameters
`name` of type `DOMString`
The name for this declaration.
No Return Value
No Exceptions

`setNamespaceURI`
Set the namespace URI for this declaration.
Parameters
`namespaceURI` of type `DOMString`
The namespace URI for this declaration.
No Return Value
No Exceptions

`setPublicId`
Set publicId for this entity
Parameters
`publicId` of type `DOMString`
The string representing the public identifier for this notation declaration, if present; `null` otherwise.
No Return Value
No Exceptions

`setRawname`
Set the rawname for this declaration.
Parameters
`rawname` of type `DOMString`
The rawname of type `Name` for this declaration.
No Return Value
No Exceptions

`setSystemId`
Set systemId for this entity
Parameters
`systemId` of type `DOMString`
The URI reference representing the system identifier for the notation declaration, if present, `null` otherwise.
No Return Value
No Exceptions

1.4. Validation and Other Interfaces

This section contains "Validation and Other" interfaces common to "AS-READ", "AS-EDIT" and "AS-DOC" parts.

Exception *ASException*

Abstract Schemas operations may throw a `ASException` [p.48] as described in their descriptions.

IDL Definition

```
exception ASException {
  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;
const unsigned short      INVALID_CHARACTER_ERR          = 5;
const unsigned short      VALIDATION_ERR                 = 6;
const unsigned short      ACTIVEAS_DELETION_ERR          = 7;
```

Definition group *ASExceptionCode*

An integer indicating the type of error generated.

Defined Constants
`ACTIVEAS_DELETION_ERR`
Raised if boundASModels is being set or removed and the activeASModel is not one of them.

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

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

`NO_AS_AVAILABLE`
: If the `DocumentEditAS` [p.53] related to the node does not have any active `ASModel` [p.25] 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.28] nor an `ASElementDecl` [p.28] .

`VALIDATION_ERR`
: Raised if document is invalid.

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

Interface *DocumentAS*

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

IDL Definition

```
interface DocumentAS : Document {
           attribute ASModel         activeASModel;
           attribute ASObjectList    boundASModels;
  ASModel            getInternalAS();
  void               setInternalAS(in ASModel as)
                                       raises(DOMException);
  void               addAS(in ASModel as);
  void               removeAS(in ASModel as)
                                       raises(ASException);
  ASElementDecl      getElementDecl(in Element node)
                                       raises(DOMException);
  ASAttributeDecl    getAttributeDecl(in Attr node)
                                       raises(DOMException);
  ASEntityDecl       getEntityDecl(in Entity node)
                                       raises(DOMException);
  ASNotationDecl     getNotationDecl(in Notation node)
                                       raises(DOMException);
  void               validate()
                                       raises(ASException);
};
```

Attributes

`activeASModel` of type `ASModel` [p.25]
: The active external `ASModel` [p.25] . Validation is responsible for not only validating the document instance against the active external `ASModel` but also for consulting the internal `ASModel`, so if an attribute is declared in the internal `ASModel` and the corresponding `ownerElements` points to a `ASElementDecl` [p.28] s defined in the active external

`ASModel`, changing the active external `ASModel` will cause the `ownerElements` to be recomputed during the validation of the document instance. If the `ownerElements` is not defined in the newly active external `ASModel`, the `ownerElements` will be an empty object list.

`boundASModels` of type `ASObjectList` [p.23]

A list of `ASObject` [p.19] s of type `SCHEMA_MODEL`s associated with a document. The `addAS` method associates an `ASModel` [p.25] with a document. An exception `ACTIVEAS_DELETION_ERR` [p.48] is thrown if the `activeASModel` is not one of the `boundASModels`.

Methods

`addAS`

Associate an `ASModel` [p.25] 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.25]

`ASModel` to be associated with the document.

No Return Value

No Exceptions

`getAttributeDecl`

Gets the abstract schema declaration for the attribute node.

Parameters

`node` of type `Attr`

The `Attr` node for which attribute declaration is to be retrieved.

Return Value

`ASAttributeDecl` [p.29]	An attribute declaration if available overwise `null`.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if no `ASModel` [p.25] is attached to the document.

`getElementDecl`

Gets the abstract schema declaration for the element node.

Parameters

`node` of type `Element`

The `Element` node for which element declaration is to be retrieved.

Return Value

`ASElementDecl` [p.28]	An element declaration if available overwise `null`.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if no `ASModel` [p.25] is attached to the document.

`getEntityDecl`

Gets the abstract schema declaration for the entity node.

Parameters

`node` of type `Entity`

The `Entity` node for which notation declaration is to be retrieved.

Return Value

`ASEntityDecl` [p.30]	A entity declaration if available overwise `null`.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if no `ASModel` [p.25] is attached to the document.

`getInternalAS`

Retrieve the internal `ASModel` [p.25] of a document.

Return Value

`ASModel` [p.25]	`ASModel`.

No Parameters

No Exceptions

`getNotationDecl`

Gets the abstract schema declaration for the notation node.

Parameters

`node` of type `Notation`

The `Notation` node for which notation declaration is to be retrieved.

Return Value

`ASNotationDecl` [p.30]	A notation declaration if available overwise `null`.

Exceptions

`DOMException`	NOT_FOUND_ERR: Raised if no `ASModel` [p.25] is attached to the document.

`removeAS`

Removes an `ASModel` [p.25] 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.25]

The `ASModel` to be removed.

Exceptions

`ASException` [p.48]	ACTIVEAS_DELETION_ERR: Raised if removing boundASModels and the activeASModel is not one of them.

No Return Value

`setInternalAS`

Sets the internal subset `ASModel` [p.25] of a document. This could be `null` as a mechanism for "removal".

Parameters

`as` of type `ASModel` [p.25]

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

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if implementation doesn’t support AS-editing.

No Return Value

`validate`

Validates the document against the `ASModel` [p.25] . If the document is mutated during validation, a warning will be issued.

Exceptions

`ASException` [p.48]	VALIDATION_ERR: Raised if an error occurs when the document is being validated against the abstract schema.

No Parameters

No Return Value

Interface *DOMImplementationAS*

This interface allows creation of an `ASWModel` [p.32] . It extends the `DOMImplementation` interface. An object that implements `DOMImplementationAS` is obtained by doing a binding specific cast from `DOMImplementation` to `DOMImplementationAS`.

IDL Definition

```
interface DOMImplementationAS : DOMImplementation {
  ASWModel           createASWModel(in boolean isNamespaceAware,
                                    in boolean container,
                                    in DOMString schemaType);
};
```

Methods

`createASWModel`

Creates an `ASWModel` [p.32] .

Parameters

`isNamespaceAware` of type `boolean`

Allow creation of `ASWModel` [p.32] with this attribute set to a specific value.

`container` of type `boolean`

Specifies that `ASWModel` [p.32] serves as a container for other `ASWModels`.

`schemaType` of type `DOMString`

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

Return Value

`ASWModel` [p.32]	An `ASWModel`.

No Exceptions

1.5. 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.54] 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` [p.49] , 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.

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.52] 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 : Node {

  // 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(ASException);
};
```

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.164] . 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 `Node::insertBefore` operation would make this document not partially valid with respect to the currently active AS.
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.49] 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

`ASException` [p.48]	NO_AS_AVAILABLE: Exception is raised if the DocumentEditAS related to this node does not have any active `ASWModel` [p.32] 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.

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` nodes 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` nodes 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 `elemTypeName` 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 `elemTypeName` 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.54] 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

1.6. 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.163] 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.

1.7. 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.

1.8. 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.

1.9. 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.

1.10. Load and Save for Abstract Schemas

The module extends the Document Object Model Load and Save [p.69] module to permit to load a `Document` using a specific `ASWModel` [p.32] and to load an `ASWModel` from an URI or `DOMInputSource` [p.90] .

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "LS-AS" 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 *ASDOMBuilder*

An Abstract Schema parser interface.

`ASDOMBuilder` provides an API for parsing Abstract Schemas and building the corresponding `ASWModel` [p.32] tree. The actual ASDOMBuilder can be created by appropriately casting the object created by an implementation that supports AS.

IDL Definition

```
interface ASDOMBuilder : ls::DOMBuilder {
           attribute ASWModel        abstractSchema;
  ASWModel           parseASURI(in DOMString uri, 
                                in DOMString schemaType)
                                        raises(ASException, 
                                               DOMSystemException);
  ASWModel           parseASInputSource(in ls::DOMInputSource is, 
                                        in DOMString schemaType)
                                        raises(ASException, 
                                               DOMSystemException);
};
```

Attributes

`abstractSchema` of type `ASWModel` [p.32]

The one active `ASWModel` [p.32] associated with a document instance. Note that the parser should set the one active `ASWModel`.

Methods

`parseASInputSource`

Parse a Abstract Schema from a location identified by an `DOMInputSource` [p.90] .

Parameters

`is` of type `ls::DOMInputSource`

The `DOMInputSource` [p.90] from which the source Abstract Schema is to be read.

`schemaType` of type `DOMString`

An absolute URI representing the type of the schema language or null if the implementation can infer a schema type. Note: For W3C XML Schema [XML Schema Part 1], applications must use the value "http://www.w3.org/2001/XMLSchema". For XML DTD [XML 1.0], applications must use the value "http://www.w3.org/TR/REC-xml". Other Schema languages are outside the scope of the W3C and therefore should recommend an absolute URI in order to use this method.

Return Value

`ASWModel` [p.32]	The newly created `ASWModel`.

Exceptions

`ASException` [p.48]	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 `ASException` [p.48] 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 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.

Parameters

`uri` of type `DOMString`

The location of the Abstract Schema to be read.

`schemaType` of type `DOMString`

An absolute URI representing the type of the schema language or null if the implementation can infer a schema type. Note: For W3C XML Schema [XML Schema Part 1], applications must use the value "http://www.w3.org/2001/XMLSchema". For XML DTD [XML 1.0], applications must use the value "http://www.w3.org/TR/REC-xml". Other Schema languages are outside the scope of the W3C and therefore should recommend an absolute URI in order to use this method.

Return Value

`ASWModel` [p.32]	The newly created `Abstract Schema`.

Exceptions

`ASException` [p.48]	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 `ASException` [p.48] 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.

Interface *ASDOMWriter*

A Abstract Schema serialization interface.

ASDOMWriters 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.

ASDOMWriter 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.103]).

IDL Definition

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

Methods

`writeASModel`

Write out the specified Abstract Schema to the specified destination.

Parameters

`destination` of type `DOMOutputStream`

The destination for the data to be written.

`model` of type `ASModel` [p.25]

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.163] 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 Features

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

Issue LS-Issue-90:
: The interaction and relationships between all the DOMBuilder and DOMWriter features need to be defined, i.e. setting x will set y and unset z.

Issue LS-Issue-91:
: DOMBuilder.entityResolver: The description should describe what support a builder is expected to provide if the resolver is not specified. When a new builder is created, should a default resolver be exposed via this attribute, to allow client code to "wrap" a basic resolver, or should the default value be null? (This kind of information would be helpful for many attributes in the DOM spec.)

Issue LS-Issue-92:
: DOMBuilder.errorHandler: When a new builder is created, should a error handler be exposed via this attribute, to allow client code to "wrap" a handler, or should the default value be null?

Issue LS-Issue-93:
: DOMBuilderFilter.whatToShow: The description of this attribute states that attribute nodes will never be passed to the filter, and the description of the filter interface also states that the document element will not be passed to the filter. What about the Document, DocumentType, Notation, and Entity nodes?

Issue LS-Issue-94:
: DocumentLS.saveXML: Why would the return value ever be null?

Issue LS-Issue-95:
: The DOMBuilder supports a "feature" called "create-entity-nodes"; is there a reason to also define "create-notation-nodes"? There’s definately less need to provide a filter of this sort. Perhaps there

should be an option to not build the DocumentType node at all, even if present?
"processing-instructions" ?

Issue LS-Issue-96:

The description of serializing character data and attributes is at variance with XML C14N rules; it seems preferable to stay consistent with C14N where possible, or at least to better motivate any departures.
For example, the description:
"Attributes containing quotes but no apostrophes are serialized in apostrophes (single quotes). Attributes containing both forms of quotes are serialized in quotes, with quotes within the value represented by the predefined entity "."
varies from C14N which never uses single quotes but always replaces a quotation mark in the attribute value with ".
Somebody should carefully review this text with respect to C14N rules, and either use C14N rules or provide feature options on DOMWriter that allows the user of DOMWriter to choose the appropriate serialization.

Issue LS-Issue-97:

Under the description of DOMWriter appears the following:
" When serializing a document the DOMWriter checks to see if the document element in the document is a DOM Level 1 element or a DOM Level 2 (or higher) element (this check is done by looking at the localName of the root element). If the root element is a DOM Level 1 element then the DOMWriter will issue an error if a DOM Level 2 (or higher) element is found while serializing. Likewise if the document element is a DOM Level 2 (or higher) element and the DOMWriter sees a DOM Level 1 element an error is issued. Mixing DOM Level 1 elements with DOM Level 2 (or higher) is not supported."
I'm not sure what this is saying. Is it describing a scenario where multiple implementations are simultaneously used with a single API and a document which was instantiated by a Level 1 implementation has an element which was instantiated by a Level 2 implementation? Wouldn't it be an error to import a Level 2 node into a Level 1 document in the first place? Or wouldn't such an import effectively downcast that Level 2 node to its Level 1 counterpart?
If, on the othe hand, this language is not talking about multiple implementations, then how is it possible to have a Level 2 implementation create a Level 1 element? Any element created by a Level 2 implementation will be a Level 2 element.

Issue LS-Issue-98:

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

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

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

Issue LS-Issue-99:

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

Issue LS-Issue-100:

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

`<foo/><bar/>`

and

`foobar`

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

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

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

Issue LS-Issue-102:

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

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.

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

Issue LS-Issue-5:

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

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

Issue LS-Issue-6:

The relationship between SAXExceptions and DOM exceptions seems confusing.

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

Issue LS-Issue-7:

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

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

Issue LS-Issue-8:

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

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

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

Issue LS-Issue-9:

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

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

The reasons for using the SAX interfaces are that they are well designed, widely implemented and used, and provide what is needed. Designing something new would create confusion for application developers (which should be used?) and make extra work for implementers of the DOM, most of whom probably already provide SAX, all for no real gain.
Resolution: Problem is gone. We are not using SAX2. The design will borrow features and concepts from SAX2 when it makes sense to do so.

Issue LS-Issue-10:
Error Reporting. Loading will be reporting well-formedness and validation errors, just like AS. A common error reporting mechanism needs to be developed.
Resolution: Resolved, see errors.html

Issue LS-Issue-11:
Another Error Reporting Question. We decided at the June f2f that validity errors should not be exceptions. This means that a document load operation could encounter multiple errors. Should these be collected and delivered as some sort of collection at the (otherwise) successful completion of the load, or should there be some sort of callback? Callbacks are harder for applications to deal with.
Resolution: Provide a callback mechanism. Provide a default error handler that throws an exception and stops further processing. From July f2f.

Issue LS-Issue-12:
Definition of "Non-validating". Exactly how much processing is done by "non-validating" parsers is not fully defined by the XML specification. In particular, they are not required to read any external entities, but are not prohibited from doing so.
Another common user request: a mode that completely ignores DTDs, both and external. Such a parser would not conform to XML 1.0, however.
For the documents produced by a non-validating load to be the same, we need to tie down exactly what processing must be done. The XML Core WG also has question as an open issue .
Some discussion is at http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000JanMar/0192.html
Here is proposal: Have three classes of parsers

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

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

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

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

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

Issue LS-Issue-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` interface in the AS spec

Issue LS-Issue-17:

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

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

Issue LS-Issue-18:

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

Resolution: Done

Issue LS-Issue-19:

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

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

Resolution: Opaque.

Issue LS-Issue-20:

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

Resolution: nobody seems to remember this, no action taken

Issue LS-Issue-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 `supported-mediatypes-only` feature on `DOMBuilder` [p.97] .

Issue LS-Issue-33:

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

This means that the suggested W3C model of normalization on serialization (early normalization) may not work, and that the receiver of the data may need to normalize it again, just in case.

Resolution: The scenario described is a quality-of-implementation issue. A transcoder converting from the one of the troublesome code pages to a Unicode representation should be responsible for re-normalizing the output.

Issue LS-Issue-34:

Features 2.1.4.1, 2 - XML Fragment Support. Should these be dropped?

Resolution: The DOM WG decided to drop support for XML fragment loading in the DOM Level 3 Load-Save module due to lack of time to define the behavior in all the edge cases, future versions of this spec might address this issue.

Issue LS-Issue-35:

XPath based document load filter. It would be plausible to have a partial (filtered) document load based on selecting the portion of the document to load with an XPath expression. This facility could be in addition to the node-by-node filtering currently specified. Or we could drop the existing filter. Implementing an XPath based selective load would require that there be an XPath processor present in addition to the parser itself.

Resolution: The DOM Level 3 spec will not define an interface for doing XPath/XPointer type filtering, implementations are free to implement XPath/XPointer based filters on top of a DOMBuilderFilter.

Issue LS-Issue-36:

MIME Type checking for DOMASBuilder.

What MIME Type checking needs to be done for parsing schemas

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

Issue LS-Issue-37:

Internal ASModel serialization for DOMWriter.

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

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

Issue LS-Issue-38:

Attribute Normalization.

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

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

Issue LS-Issue-39:

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

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

Issue LS-Issue-40:

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

Resolution: Yes, the description is acceptable.

Issue LS-Issue-41:

Do we need filter support in DOMWriter too?

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

Issue LS-Issue-42:

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

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

Issue LS-Issue-43:

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

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

Issue LS-Issue-44:

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

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

Issue LS-Issue-45:

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

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

Issue LS-Issue-46:

Should supporting async/sync loading be optional?

Resolution: Yes.

Issue LS-Issue-47:

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

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

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

Issue LS-Issue-48:

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

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

Issue LS-Issue-49:

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

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

Issue LS-Issue-50:

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

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

Issue LS-Issue-51:

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

Resolution: Yes, supportsFeature removed.

Issue LS-Issue-52:

Is the API dependencies on the Events spec acceptable?

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

Issue LS-ISSUE-53:

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

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

Issue LS-Issue-54:

"canonical-form" needs a correct reference to the spec for canonical XML.

Issue LS-Issue-55:

How should default attributes be dealt with wrt DOMBuilderFilter?

Resolution: All default content must be passed to the filter.

Issue LS-Issue-56:

Should we make it possible to `SKIP` an element in `DOMBuilderFilter::acceptNode`?

Resolution: Yes, done.

Issue LS-Issue-57:

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

Resolution: [DOM Level 2 Core] allows empty strings as a real namespace URI. If the `namespaceURI` of a `Node` is empty string, the serialization will treat them as `null`, ignoring the prefix if any.

Issue LS-Issue-59:

ACTION_APPEND is confusing, can we clarify it?

Resolution: make it ACTION_APPEND_AS_CHILDREN (2002-01-28)

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

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

Issue LS-Issue-62:
createDOMBuilder: If MODE_SYNCHRONOUS and MODE_ASYNCHRONOUS are the only anticipated values, then a boolean parameter would be preferred. If it stays a unsigned short, then there needs to be a exception for unrecognized values.
Resolution: We keep the unsigned hsort for future possible extension.
"NOT_SUPPORTED_ERR: Raised if the requested mode is not supported."

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

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

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

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

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

Issue LS-Issue-68:

DOMBuilder.parseWithContext: Should throw DOMSystemExceptions. Should throw NO_MODIFICATION_ALLOWED_ERR if context node (or parent) is read-only. Returning the created node would be desirable.

Resolution: Exception added. But the created node can’t be returned since there might be more than one node created.

Issue LS-Issue-69:

How does DOMBuilder.parseWithContext interact with any event listeners registered on the context node or its ancestors?

Resolution: Description on what mutation events are fired when using parseWithContext() added.

Issue LS-Issue-70:

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

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

Issue LS-Issue-71:

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

Issue LS-Issue-72:

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

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

Issue LS-Issue-73:

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

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

Issue LS-Issue-74:

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

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

Issue LS-Issue-75:

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

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

Issue LS-Issue-76:

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

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

Issue LS-Issue-77:

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

Resolution: See issue 90.

Issue LS-Issue-78:

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

Resolution: Description updated.

Issue LS-Issue-79:

DOMWriter.writeToString method: How is this affected by encoding? It will be represented internally as UTF-16 on most binding, but users who have set encoding to ISO-8859-1 or US-ASCII might expect no code points higher than 255 or 127 respectively so they can naively write out the string to a file later.

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

Issue LS-Issue-80:

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

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

Issue LS-Issue-81:

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

Resolution: Description updated.

Issue LS-Issue-82:

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

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

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

Issue LS-Issue-83:

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

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

Issue LS-Issue-84:

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

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

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

Issue LS-Issue-85:

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

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

Issue LS-Issue-86:

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

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

Issue LS-Issue-87:

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

Resolution:

Issue LS-Issue-88:

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

Resolution: Error handler methods can not throw exceptions. The main reason for this is that in the async loading case there's noone on the receiving end of the call to the error handler that would be able to deal with the exception. And besides, exceptions are for exceptional cases, this would not be such a case.

Issue LS-Issue-89:

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

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

Issue LS-Issue-101:

reconsider your removal of the namespaces feature

Resolution: added back in the draft.

2.3. Interfaces

This section defines an *API* [p.163] for loading (parsing) XML documents [XML 1.0] 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.86] -- A new `DOMImplementation` interface that provides the factory methods for creating the objects required for loading and saving.
- `DOMBuilder` [p.97] -- A parser interface.
- `DOMInputSource` [p.90] -- Encapsulate information about the XML document to be loaded.
- `DOMEntityResolver` [p.93] -- During loading, provides a way for applications to redirect references to external entities.
- `DOMBuilderFilter` [p.94] -- Provide the ability to examine and optionally remove Element nodes as they are being processed during the parsing of a document.
- `DOMWriter` [p.103] -- An interface for writing out or serializing DOM documents.
- `DocumentLS` [p.88] -- Provides a client or browser style interface for loading and saving.
- `ParseErrorEvent` [p.96] -- ParseErrorEvent is the event that is fired if there’s an error in the XML document being parsed using the methods of DocumentLS.

2.3.1. Fundamental interface

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

Interface *DOMImplementationLS*

`DOMImplementationLS` contains the factory methods for creating objects that implement the `DOMBuilder` [p.97] (parser) and `DOMWriter` [p.103] (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,
                                      in DOMString schemaType)
                                        raises(DOMException);
  DOMWriter          createDOMWriter();
  DOMInputSource     createDOMInputSource();
};
```

Definition group *DOMIMplementationLSMode*

An integer indicating which type of mode this is.

Defined Constants

`MODE_ASYNCHRONOUS`
: Create an asynchronous `DOMBuilder` [p.97] .

`MODE_SYNCHRONOUS`
: Create a synchronous `DOMBuilder` [p.97] .

Methods

`createDOMBuilder`

Create a new `DOMBuilder` [p.97] . 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.97] that is created will operate in synchronous mode, if it’s `MODE_ASYNCHRONOUS` then the `DOMBuilder` that is created will operate in asynchronous mode.

`schemaType` of type `DOMString`
: An absolute URI representing the type of the schema language used during the load of a `Document` using the newly created `DOMBuilder` [p.97] . Note that no lexical checking is done on the absolute URI. In order to create a `DOMBuilder` for any kind of schema types (i.e. the DOMBuilder will be free to use any schema found), use the value `null`.

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

Return Value

`DOMBuilder` [p.97]	The newly created `DOMBuilder` object. This `DOMBuilder` is either synchronous or asynchronous depending on the value of the `mode` argument.

Exceptions

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

`createDOMInputSource`

Create a new "empty" `DOMInputSource` [p.90] .

Return Value

`DOMInputSource` [p.90]	The newly created `DOMInputSource` object.

No Parameters
No Exceptions

`createDOMWriter`

Create a new `DOMWriter` [p.103] object. `DOMWriter`s are used to serialize a DOM tree back into an XML document.

Return Value

`DOMWriter` [p.103]	The newly created `DOMWriter` object.

No Parameters
No Exceptions

Interface *DocumentLS*

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

uses the default features.

IDL Definition

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

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

Attributes

`async` of type `boolean`

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

Issue async-1:

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

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

Exceptions on setting

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

Methods

`abort`

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

No Parameters

No Return Value

No Exceptions

`load`

Replaces the content of the document with the result of parsing the given URI. Invoking this method will either block the caller or return to the caller immediately depending on the value of the async attribute. Once the document is fully loaded the document will fire a "load" event that the caller can register as a listener for. If an error occurs the document will fire an "error" event so that the caller knows that the load failed (see `ParseErrorEvent` [p.96]). If this method is called on a document that is currently loading, the current load is interrupted and the new URI load is initiated.

Parameters

`uri` of type `DOMString`

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

Return Value

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

No Exceptions

`loadXML`

Replace the content of the document with the result of parsing the input string, this method is always synchronous. This method always parses from a DOMString, which means the data is always UTF16. All other encoding information is ignored.

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.

2.3.2. Load Interfaces

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

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

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

Interface *DOMInputSource*

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

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

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

There are two places that the application will deliver this input source to the parser: as the argument to the `parse` method, or as the return value of the `DOMEntityResolver.resolveEntity` [p.93] method.

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

The `DOMBuilder` [p.97] will use the `DOMInputSource` object to determine how to read XML input. If there is a character stream available, the parser will read that stream directly; if not, the parser will use a byte stream, if available; if neither a character stream nor a byte stream is available, the parser will attempt to open a URI connection to the resource identified by the system identifier.

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

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

IDL Definition

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

Attributes

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

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

`characterStream` of type `DOMReader`
: An attribute of a language-binding dependent type that represents a stream of *16-bit units.* [p.163] Application must encode the stream using UTF-16 (defined in [Unicode 2.0] and Amendment 1 of [ISO/IEC 10646]).
If a character stream is specified, the parser will ignore any byte stream and will not attempt to open a URI connection to the system identifier.

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

encoding specified in the XML claration or the Text Declaration, or an encoding obtained from a higher level protocol, such as HTTP [IETF RFC 2616].

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

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

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

Interface *LSLoadEvent*

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

IDL Definition

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

Attributes

`inputSource` of type `DOMInputSource` [p.90] , readonly
: The input source that was parsed.

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

Interface *LSProgressEvent*

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

IDL Definition

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

Attributes

`inputSource` of type `DOMInputSource` [p.90] , readonly
: The input source that is being parsed.

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

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

Interface *DOMEntityResolver*

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

Applications needing to implement customized handling for external entities must implement this interface and register their implementation by setting the `entityResolver` attribute of the `DOMBuilder` [p.97] .

The `DOMBuilder` [p.97] will then allow the application to intercept any external entities (including the external DTD subset and external parameter entities) before including them.

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

Note: `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.97] 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.97] must resolve it fully before reporting it to the application through this interface.
(***ED:*** See issue #4. An alternative would be to pass the URI out without resolving it, and to

provide a base as an additional parameter. SAX resolves URIs first, and does not provide a base.)

Parameters

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

`systemId` of type `DOMString`
: The system identifier, a URI reference [IETF RFC 2396], of the external entity being referenced exactly as written in the source.

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

Return Value

`DOMInputSource` [p.90]	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.

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

Any exception raised in the filter are ignored by the `DOMBuilder` [p.97] .

The constants `FILTER_ACCEPT`, `FILTER_REJECT` and `FILTER_SKIP` are defined in DOM Level 2 Traversal [DOM Level 2 Traversal and Range].

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

IDL Definition

```
interface DOMBuilderFilter {
  const unsigned short      FILTER_INTERRUPT               = 4;
  unsigned short     startContainer(in Node snode);
  unsigned short     acceptNode(in Node enode);
  readonly attribute unsigned long   whatToShow;
};
```

Constant *FILTER_INTERRUPT*

Interrupt the normal processing of the document.

Attributes

`whatToShow` of type `unsigned long`, readonly

Tells the `DOMBuilder` [p.97] what types of nodes to show to the filter. See `NodeFilter` for definition of the constants. The constant `SHOW_ATTRIBUTE` is meaningless here, attribute nodes will never be passed to a `DOMBuilderFilter`.

Methods

`acceptNode`

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

Parameters

`enode` of type `Node`

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

Return Value

`unsigned short`	• `FILTER_ACCEPT` if this `Node` should be included in the DOM document being built. • `FILTER_REJECT` if the `Node` and all of its children should be rejected. • `FILTER_SKIP` if the `Node` should be skipped and the `Node` should be replaced by all the children of the `Node`.

No Exceptions

`startContainer`

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 `startContainer` function.

The element node passed to `startContainer` 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 `startContainer` 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 startContainer?

Return Value

`unsigned short`	• `FILTER_ACCEPT` if this `Element` should be included in the DOM document being built. • `FILTER_REJECT` if the `Element` and all of its children should be rejected. • `FILTER_SKIP` if the `Element` should be rejected. All of its children are inserted in place of the rejected `Element` node. • `FILTER_INTERRUPT` if the filter wants to stop the processing of the document. Interrupting the processing of the document does no longer garantee that the entire is XML well-formed. Returning any other values will result in unspecified behavior.

No Exceptions

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.

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.86] interface by invoking its `createDOMBuilder`method.

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

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

  Issue Infoset:

  XML Schemas does not modify the XML attribute normalization but represents their normalized value in an other information item property: [schema normalized value]
  Resolution: XML Schema normalization only occurs if `datatype-normalization` is set to `true`.

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

Events supported by asynchronous `DOMBuilder` are:

- **load**: The document that’s being loaded is completely parsed, see the definition of `LSLoadEvent` [p.92]
- **progress**: Progress notification, see the definition of `LSProgressEvent` [p.92]

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

`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:

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

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

`"charset-overrides-xml-encoding"`
: **`true`**
  : [*required*] (*default*)
    If a higher level protocol such as HTTP [IETF RFC 2616] provides an indication of the character encoding of the input stream being processed, that will override any encoding specified in the XML declaration or the Text declaration (see also [XML 1.0] 4.3.3 "Character Encoding in Entities"). Explicitly setting an encoding in the `DOMInputSource` [p.90] overrides encodings from the protocol.

  `false`
  : [*required*]
    Any character set encoding information from higher level protocols is ignored by the parser.

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

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

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

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

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

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

`"supported-mediatypes-only"`

  `true`

    [*optional*]
    Check that the media type of the parsed resource is a supported media type and call the error handler if an unsupported media type is encountered. The media types defined in [IETF RFC 3023] must be accepted.

  `false`

    [*required*] (*default*)
    Don’t check the media type, accept any type of data.

`"validate-if-schema"`

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

`"validation"`

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

`"whitespace-in-element-content"`

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

IDL Definition

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

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

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

Definition group *ACTION_TYPES*

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

Defined Constants

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

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

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

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

Attributes

`entityResolver` of type `DOMEntityResolver` [p.93]
: If a `DOMEntityResolver` [p.93] 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 `DOMDocumentBuilder` will call back to the `errorHandler` with the error information. When the document loading process calls the error handler the node closest to where the error occured is passed to the error handler, if the implementation is unable to pass the node where the error occures the document Node is passed to the error handler. In addition to passing the Node closest to to where the error occured, the implementation should also pass any other valuable information to the error handler, such as file name, line number, and so on. Mutations to the document from within an error handler will result in implementation dependent behavour.

`filter` of type `DOMBuilderFilter` [p.94]
: When the application provides a filter, the parser will call out to the filter at the completion of the construction of each `Element` node. The filter implementation can choose to remove the element from the document being constructed (unless the element is the document element) or to terminate the parse early. If the document is being validated when it’s loaded the validation happens before the filter is called.

Methods

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

Parameters

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

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

Return Value

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

No Exceptions

`getFeature`

Look up the value of a feature.

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

Parameters

`name` of type `DOMString`

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

Return Value

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

Exceptions

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

`parse`

Parse an XML document from a resource identified by a `DOMInputSource` [p.90] .

Parameters

`is` of type `DOMInputSource` [p.90]

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

Parameters

`uri` of type `DOMString`

The location of the XML document to be read.

Return Value

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

No Exceptions

`parseWithContext`

Parse an XML document or fragment from a resource identified by a `DOMInputSource` [p.90] and insert the content into an existing document at the position specified with the `contextNode` and `action` arguments. When parsing the input stream the context node is used for resolving unbound namespace prefixes.

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

If an error occurs while parsing, the caller is notified through the error handler.

Parameters

`is` of type `DOMInputSource` [p.90]

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. This node must be a `Document` node, a `DocumentFragment` node, or a node of a type that is allowed as a child of an element, e.g. it can not be an attribute node.

`action` of type `unsigned short`

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

Exceptions

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

No Return Value

`setFeature`

Set the state of a feature.
The feature name has the same form as a DOM hasFeature string.
It is possible for a `DOMBuilder` to recognize a feature name but to be unable to set its value.

Parameters

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

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

Exceptions

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

No Return Value

2.3.3. Save Interface

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

Interface *DOMWriter*

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

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

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

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

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

Note: The serialization of a DOM Node does not always generate a *well-formed* [p.164] XML document, i.e. a `DOMBuilder` [p.97] might through fatal errors when parsing the resulting serialization.

Within the character data of a document (outside of markup), any characters that cannot be represented directly are replaced with character references. Occurrences of ’<’ and ’&’ are replaced by the predefined entities &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 serializing a document is a combination of the algorithms used for lookupNamespaceURI and lookupNamespacePrefix .

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

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

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

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

`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 features that must be recognized by all implementations.

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

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

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

`"canonical-form"`

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

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

`"format-pretty-print"`

`true`
: [*optional*]
: Formatting the output by adding whitespace to produce a pretty-printed, indented,

human-readable form. The exact form of the transformations is not specified by this specification. Setting this feature to true will set the feature "canonical-form" to false.

`false`

[*required*] (*default*)

Don’t pretty-print the result.

`"normalize-characters"`

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

`"split-cdata-sections"`

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

`"validation"`

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

`"whitespace-in-element-content"`

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

IDL Definition

```
interface DOMWriter {
  void               setFeature(in DOMString name,
                                in boolean state)
                                        raises(DOMException);
  boolean            canSetFeature(in DOMString name,
                                   in boolean state);
  boolean            getFeature(in DOMString name)
                                        raises(DOMException);
           attribute DOMString       encoding;
           attribute DOMString       newLine;
           attribute DOMWriterFilter filter;
           attribute DOMErrorHandler errorHandler;
  boolean            writeNode(in DOMOutputStream destination,
                               in Node wnode)
                                        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, or the owner document of the item, specifies an encoding (i.e. the "actualEncoding" from the document) specified encoding, that value will be used.
- If neither of the above provides an encoding name, a default encoding of "UTF-8" will be used.

The default value is `null`.

`errorHandler` of type `DOMErrorHandler`

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

`filter` of type `DOMWriterFilter` [p.109]

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

`newLine` of type `DOMString`

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

`null`

Use a default end-of-line sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by [XML 1.0] 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`	NOT_FOUND_ERR: Raised when the `DOMWriter` does not recognize the feature name.

`setFeature`

Set the state of a feature.

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

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

Parameters

`name` of type `DOMString`

The feature name.

`state` of type `boolean`

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

Exceptions

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

No Return Value

`writeNode`

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

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

Parameters

`destination` of type `DOMOutputStream`

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 DOMString (this method completely ignores all the encoding information avaliable). Writing a Document or Entity node produces a serialized form that is well formed XML. Writing other node types produces a fragment of text in a form that is not fully defined by this document, but that should be useful to a human for debugging or diagnostic purposes.
Error handler is called if encoding not supported...

Parameters

`wnode` of type `Node`

The node to be written.

Return Value

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

Exceptions

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

Interface *DOMWriterFilter*

`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.103] 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`.

Appendix A: IDL Definitions

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

The IDL files are also available as:
http://www.w3.org/TR/2002/WD-DOM-Level-3-ASLS-20020409/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::Document Document;
  typedef dom::Element Element;
  typedef dom::Attr Attr;
  typedef dom::Entity Entity;
  typedef dom::Notation Notation;
  typedef dom::DOMImplementation DOMImplementation;
  typedef dom::Node Node;
  typedef dom::NodeList NodeList;
  typedef dom::DOMOutputStream DOMOutputStream;

  interface ASModel;
  interface ASElementDecl;
  interface ASAttributeDecl;
  interface ASEntityDecl;
  interface ASNotationDecl;
  interface ASWElementDecl;
  interface ASWAttributeDecl;
  interface ASWNotationDecl;
  interface ASWEntityDecl;
  interface ASWContentModel;
  interface NodeEditAS;

  exception ASException {
    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;
  const unsigned short      INVALID_CHARACTER_ERR          = 5;
  const unsigned short      VALIDATION_ERR                 = 6;
  const unsigned short      ACTIVEAS_DELETION_ERR          = 7;


  interface ASConstants {

    // ASObject Types
    const unsigned short      ELEMENT_DECLARATION            = 1;
    const unsigned short      ATTRIBUTE_DECLARATION          = 2;
    const unsigned short      NOTATION_DECLARATION           = 3;
    const unsigned short      ENTITY_DECLARATION             = 4;
    const unsigned short      CONTENT_MODEL                  = 5;
    const unsigned short      SCHEMA_MODEL                   = 6;


    // Schema Model types
    const unsigned short      INTERNAL_SUBSET                = 30;
    const unsigned short      EXTERNAL_SUBSET                = 31;
    const unsigned short      NOT_USED                       = 32;


    // Entity Types
    const unsigned short      INTERNAL_ENTITY                = 33;
    const unsigned short      EXTERNAL_ENTITY                = 34;


    // Content Model Types
    const unsigned short      EMPTY_CONTENTTYPE              = 40;
    const unsigned short      SIMPLE_CONTENTTYPE             = 41;
    const unsigned short      ELEMENT_CONTENTTYPE            = 42;
    const unsigned short      MIXED_CONTENTTYPE              = 43;
    const unsigned short      ANY_CONTENTTYPE                = 44;


    // Content model compositors
    const unsigned short      SEQUENCE_CM                    = 50;
    const unsigned short      CHOICE_CM                      = 51;
    const unsigned short      ALL_CM                         = 52;
    const unsigned short      UNDEFINED_CM                   = 53;
    const unsigned short      ATOMIC_CM                      = 54;


    // Value Constraint
    const unsigned short      NONE_VC                        = 0;
    const unsigned short      DEFAULT_VC                     = 60;
    const unsigned short      FIXED_VC                       = 61;
    const unsigned short      REQUIRED_VC                    = 62;


    // Definition of unbounded
    const unsigned long       UNBOUNDED                      = MAX_VALUE;
  };

  interface ASObject {
    readonly attribute unsigned short  objectType;
    readonly attribute ASModel         ownerModel;
    readonly attribute DOMString       rawname;
    readonly attribute DOMString       name;
    readonly attribute DOMString       namespace;
    ASObject           cloneASObject(in boolean deep)
                                        raises(ASException);
  };

  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      ANYSIMPLETYPE_DATATYPE         = 216;
    const unsigned short      ANYTYPE_DATATYPE               = 216;
  };
```

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

interface ASNamedObjectMap {
  readonly attribute unsigned long   length;
  ASObject           item(in unsigned long index);
  ASObject           getNamedItem(in DOMString name);
  ASObject           getNamedItemNS(in DOMString namespaceURI,
                                    in DOMString localName);
};

interface ASModel : ASObject {
  readonly attribute boolean         namespaceAware;
  readonly attribute unsigned short  usage;
  readonly attribute DOMString       location;
  readonly attribute DOMString       hint;
  readonly attribute boolean         container;
  ASNamedObjectMap   getComponents(in unsigned short objectType);

  // Convenience method to retrive named top-level declarations

  ASElementDecl      getElementDecl(in DOMString name,
                                    in DOMString targetNamespace);
  ASAttributeDecl    getAttributeDecl(in DOMString name,
                                      in DOMString targetNamespace);
  ASEntityDecl       getEntityDecl(in DOMString name);
  ASNotationDecl     getNotationDecl(in DOMString name,
                                     in DOMString targetNamespace);
};

interface ASContentModel : ASObject {
  readonly attribute unsigned short  contentModelType;
  readonly attribute unsigned long   minOccurs;
  readonly attribute unsigned long   maxOccurs;
  readonly attribute ASObjectList    subModels;
};

interface ASElementDecl : ASObject {
  readonly attribute ASDataType      type;
  readonly attribute unsigned short  contentType;
  readonly attribute boolean         strictMixedContent;
  readonly attribute ASContentModel  contentModel;
  readonly attribute boolean         isPCDataOnly;
  readonly attribute ASNamedObjectMap attributeDecls;
  ASAttributeDecl    getAttributeDecl(in DOMString name,
                                      in DOMString targetNamespace);
};

interface ASAttributeDecl : ASObject {
  readonly attribute ASDataType      type;
  readonly attribute DOMString       enumAttr;
  readonly attribute ASObjectList    ownerElementDeclarations;
  readonly attribute unsigned short  defaultType;
  readonly attribute DOMString       value;
```

```
};

interface ASEntityDecl : ASObject {
  readonly attribute unsigned short  entityType;
  readonly attribute DOMString       entityValue;
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       publicId;
};

interface ASNotationDecl : ASObject {
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       publicId;
};

interface ASWModel : ASModel {
  void               setLocation(in DOMString location);
  void               setHint(in DOMString hint);
  void               addComponent(in ASObject declaration);
  void               removeComponent(in ASObject declaration);
  void               addASModel(in ASModel declaration);
  void               removeASModel(in ASModel declaration);
  ASObjectList       getASModels();
  ASObject           importASObject(in ASObject asobject);
  void               insertASObject(in ASObject asobject);
  boolean            validate();
  ASWElementDecl     createASWElementDecl(in DOMString namespaceURI,
                                          in DOMString name)
                                     raises(ASException);
  ASWAttributeDecl   createASWAttributeDecl(in DOMString namespaceURI,
                                            in DOMString name)
                                     raises(ASException);
  ASWNotationDecl    createASWNotationDecl(in DOMString namespaceURI,
                                           in DOMString name,
                                           in DOMString systemId,
                                           in DOMString publicId)
                                     raises(ASException);
  ASWEntityDecl      createASWEntityDecl(in DOMString name)
                                     raises(ASException);
  ASWContentModel    createASWContentModel(in DOMString name,
                                           in DOMString namespaceURI,
                                           in unsigned long minOccurs,
                                           in unsigned long maxOccurs,
                                           in unsigned short operator)
                                     raises(ASException);
};

interface ASWNamedObjectMap : ASNamedObjectMap {
  ASObject           removeNamedItem(in DOMString name)
                                     raises(ASException);
  ASObject           setNamedItem(in ASObject newASObject)
                                     raises(ASException,
                                            ASException);
  ASObject           setNamedItemNS(in ASObject arg)
                                     raises(ASException);
  ASObject           removeNamedItemNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                     raises(dom::DOMException);
```

```
};

interface ASWElementDecl : ASElementDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespace(in DOMString namespaceURI);
  void               setStrictMixedContent(in boolean mixedContent);
  void               setType(in ASDataType type);
  void               setContentType(in unsigned short contentType);
  void               setContentModel(in ASWContentModel contentModel);
  void               addAttributeDecl(in ASWAttributeDecl attributeDecl);
  ASWAttributeDecl   removeAttributeDecl(in ASWAttributeDecl attributeDecl);
};

interface ASWContentModel : ASContentModel {
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setContentModelType(in unsigned short operator);
  void               setMinOccurs(in unsigned long minOccurs);
  void               setMaxOccurs(in unsigned long maxOccurs);
  void               removeSubModel(in ASObject oldObject);
  ASObject           insertBeforeSubModel(in ASObject newObject,
                                          in ASObject refObject)
                                      raises(ASException);
  unsigned long      appendSubModel(in ASObject newObject)
                                      raises(ASException);
};

interface ASWAttributeDecl : ASAttributeDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setType(in ASDataType type);
  void               setValue(in DOMString value);
  void               setEnumAttr(in DOMString enumeration);
  void               setDefaultType(in unsigned short constraint);
};

interface ASWEntityDecl : ASEntityDecl {
  void               setRawname(in DOMString rawname);
  void               setEntityType(in unsigned short type);
  void               setEntityValue(in DOMString value);
  void               setSystemId(in DOMString systemId);
  void               setPublicId(in DOMString publicId);
};

interface ASWNotationDecl : ASNotationDecl {
  void               setRawname(in DOMString rawname);
  void               setName(in DOMString name);
  void               setNamespaceURI(in DOMString namespaceURI);
  void               setSystemId(in DOMString systemId);
  void               setPublicId(in DOMString publicId);
};

interface DocumentAS : Document {
           attribute ASModel         activeASModel;
           attribute ASObjectList    boundASModels;
  ASModel            getInternalAS();
  void               setInternalAS(in ASModel as)
                                      raises(dom::DOMException);
  void               addAS(in ASModel as);
  void               removeAS(in ASModel as)
                                      raises(ASException);
  ASElementDecl      getElementDecl(in Element node)
                                      raises(dom::DOMException);
  ASAttributeDecl    getAttributeDecl(in Attr node)
                                      raises(dom::DOMException);
  ASEntityDecl       getEntityDecl(in Entity node)
                                      raises(dom::DOMException);
  ASNotationDecl     getNotationDecl(in Notation node)
                                      raises(dom::DOMException);
  void               validate()
                                      raises(ASException);
};

interface DOMImplementationAS : DOMImplementation {
  ASWModel           createASWModel(in boolean isNamespaceAware,
                                    in boolean container,
                                    in DOMString schemaType);
};

interface DocumentEditAS : NodeEditAS {
           attribute boolean         continuousValidityChecking;
};

interface NodeEditAS : Node {

  // 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(ASException);
};

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 ASDOMBuilder : ls::DOMBuilder {
             attribute ASWModel        abstractSchema;
    ASWModel           parseASURI(in DOMString uri,
                                  in DOMString schemaType)
                                        raises(ASException,
                                               dom::DOMSystemException);
    ASWModel           parseASInputSource(in ls::DOMInputSource is,
                                          in DOMString schemaType)
                                        raises(ASException,
                                               dom::DOMSystemException);
  };

  interface ASDOMWriter : 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::DOMString DOMString;
  typedef dom::Node Node;
  typedef dom::DOMInputStream DOMInputStream;
  typedef dom::DOMReader DOMReader;
  typedef dom::DOMErrorHandler DOMErrorHandler;
  typedef dom::Document Document;
  typedef dom::DOMOutputStream DOMOutputStream;
  typedef dom::DOMError DOMError;

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

  interface DOMImplementationLS {

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

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

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

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

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

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

  interface DOMBuilderFilter {
    const unsigned short      FILTER_INTERRUPT               = 4;
    unsigned short     startContainer(in Node snode);
    unsigned short     acceptNode(in Node enode);
    readonly attribute unsigned long   whatToShow;
  };

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

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

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

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

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

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

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

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

#endif // _LS_IDL_
```

ls.idl:

Appendix B: Java Language Binding

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

The Java files are also available as
http://www.w3.org/TR/2002/WD-DOM-Level-3-ASLS-20020409/java-binding.zip

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

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

public class ASException extends RuntimeException {
    public ASException(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;
    public static final short INVALID_CHARACTER_ERR     = 5;
    public static final short VALIDATION_ERR            = 6;
    public static final short ACTIVEAS_DELETION_ERR     = 7;

}
```

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

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

public interface ASConstants {
    // ASObject Types
    public static final short ELEMENT_DECLARATION       = 1;
    public static final short ATTRIBUTE_DECLARATION     = 2;
    public static final short NOTATION_DECLARATION      = 3;
    public static final short ENTITY_DECLARATION        = 4;
    public static final short CONTENT_MODEL             = 5;
    public static final short SCHEMA_MODEL              = 6;

    // Schema Model types
    public static final short INTERNAL_SUBSET           = 30;
    public static final short EXTERNAL_SUBSET           = 31;
    public static final short NOT_USED                  = 32;

    // Entity Types
    public static final short INTERNAL_ENTITY           = 33;
    public static final short EXTERNAL_ENTITY           = 34;

    // Content Model Types
    public static final short EMPTY_CONTENTTYPE         = 40;
    public static final short SIMPLE_CONTENTTYPE        = 41;
    public static final short ELEMENT_CONTENTTYPE       = 42;
    public static final short MIXED_CONTENTTYPE         = 43;
    public static final short ANY_CONTENTTYPE           = 44;

    // Content model compositors
    public static final short SEQUENCE_CM               = 50;
    public static final short CHOICE_CM                 = 51;
    public static final short ALL_CM                    = 52;
    public static final short UNDEFINED_CM              = 53;
    public static final short ATOMIC_CM                 = 54;

    // Value Constraint
    public static final short NONE_VC                   = 0;
    public static final short DEFAULT_VC                = 60;
    public static final short FIXED_VC                  = 61;
    public static final short REQUIRED_VC               = 62;

    // Definition of unbounded
    public static final int UNBOUNDED                 = MAX_VALUE;

}
```

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

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

public interface ASObject {
    public short getObjectType();

    public ASModel getOwnerModel();

    public String getRawname();

    public String getName();

    public String getNamespace();

    public ASObject cloneASObject(boolean deep)
                                  throws ASException;

}
```

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 ANYSIMPLETYPE_DATATYPE    = 216;
    public static final short ANYTYPE_DATATYPE          = 216;

}
```

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;

public interface ASNamedObjectMap {
    public int getLength();

    public ASObject item(int index);

    public ASObject getNamedItem(String name);

    public ASObject getNamedItemNS(String namespaceURI,
                                   String localName);

}
```

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

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

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

    public short getUsage();

    public String getLocation();

    public String getHint();

    public boolean getContainer();

    public ASNamedObjectMap getComponents(short objectType);

    // Convenience method to retrive named top-level declarations

    public ASElementDecl getElementDecl(String name,
                                        String targetNamespace);

    public ASAttributeDecl getAttributeDecl(String name,
                                            String targetNamespace);

    public ASEntityDecl getEntityDecl(String name);

    public ASNotationDecl getNotationDecl(String name,
                                          String targetNamespace);

}
```

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

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

public interface ASContentModel extends ASObject {
    public short getContentModelType();
    public int getMinOccurs();

    public int getMaxOccurs();

    public ASObjectList getSubModels();

}
```

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

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

public interface ASElementDecl extends ASObject {
    public ASDataType getType();

    public short getContentType();

    public boolean getStrictMixedContent();

    public ASContentModel getContentModel();

    public boolean getIsPCDataOnly();

    public ASNamedObjectMap getAttributeDecls();

    public ASAttributeDecl getAttributeDecl(String name,
                                            String targetNamespace);

}
```

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

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

public interface ASAttributeDecl extends ASObject {
    public ASDataType getType();

    public String getEnumAttr();

    public ASObjectList getOwnerElementDeclarations();

    public short getDefaultType();

    public String getValue();

}
```

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

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

public interface ASEntityDecl extends ASObject {
    public short getEntityType();

    public String getEntityValue();
    public String getSystemId();

    public String getPublicId();

}
```

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

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

public interface ASNotationDecl extends ASObject {
    public String getSystemId();

    public String getPublicId();

}
```

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

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

public interface ASWModel extends ASModel {
    public void setLocation(String location);

    public void setHint(String hint);

    public void addComponent(ASObject declaration);

    public void removeComponent(ASObject declaration);

    public void addASModel(ASModel declaration);

    public void removeASModel(ASModel declaration);

    public ASObjectList getASModels();

    public ASObject importASObject(ASObject asobject);

    public void insertASObject(ASObject asobject);

    public boolean validate();

    public ASWElementDecl createASWElementDecl(String namespaceURI,
                                              String name)
                                              throws ASException;

    public ASWAttributeDecl createASWAttributeDecl(String namespaceURI,
                                                  String name)
                                                  throws ASException;

    public ASWNotationDecl createASWNotationDecl(String namespaceURI,
                                                String name,
                                                String systemId,
                                                String publicId)
                                                throws ASException;
    public ASWEntityDecl createASWEntityDecl(String name)
                                             throws ASException;

    public ASWContentModel createASWContentModel(String name,
                                                String namespaceURI,
                                                int minOccurs,
                                                int maxOccurs,
                                                short operator)
                                                throws ASException;

}
```

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

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

import org.w3c.dom.DOMException;

public interface ASWNamedObjectMap extends ASNamedObjectMap {
    public ASObject removeNamedItem(String name)
                                    throws ASException;

    public ASObject setNamedItem(ASObject newASObject)
                                 throws ASException, ASException;

    public ASObject setNamedItemNS(ASObject arg)
                                   throws ASException;

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

}
```

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

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

public interface ASWElementDecl extends ASElementDecl {
    public void setRawname(String rawname);

    public void setName(String name);

    public void setNamespace(String namespaceURI);

    public void setStrictMixedContent(boolean mixedContent);

    public void setType(ASDataType type);

    public void setContentType(short contentType);

    public void setContentModel(ASWContentModel contentModel);

    public void addAttributeDecl(ASWAttributeDecl attributeDecl);
    public ASWAttributeDecl removeAttributeDecl(ASWAttributeDecl attributeDecl);

}
```

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

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

public interface ASWContentModel extends ASContentModel {
    public void setName(String name);

    public void setNamespaceURI(String namespaceURI);

    public void setContentModelType(short operator);

    public void setMinOccurs(int minOccurs);

    public void setMaxOccurs(int maxOccurs);

    public void removeSubModel(ASObject oldObject);

    public ASObject insertBeforeSubModel(ASObject newObject,
                                         ASObject refObject)
                                         throws ASException;

    public int appendSubModel(ASObject newObject)
                              throws ASException;

}
```

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

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

public interface ASWAttributeDecl extends ASAttributeDecl {
    public void setRawname(String rawname);

    public void setName(String name);

    public void setNamespaceURI(String namespaceURI);

    public void setType(ASDataType type);

    public void setValue(String value);

    public void setEnumAttr(String enumeration);

    public void setDefaultType(short constraint);

}
```

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

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

public interface ASWEntityDecl extends ASEntityDecl {
    public void setRawname(String rawname);

    public void setEntityType(short type);

    public void setEntityValue(String value);

    public void setSystemId(String systemId);

    public void setPublicId(String publicId);

}
```

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

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

public interface ASWNotationDecl extends ASNotationDecl {
    public void setRawname(String rawname);

    public void setName(String name);

    public void setNamespaceURI(String namespaceURI);

    public void setSystemId(String systemId);

    public void setPublicId(String publicId);

}
```

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

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

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.DOMException;
import org.w3c.dom.Notation;
import org.w3c.dom.Attr;
import org.w3c.dom.Entity;

public interface DocumentAS extends Document {
    public ASModel getActiveASModel();
    public void setActiveASModel(ASModel activeASModel);

    public ASObjectList getBoundASModels();
    public void setBoundASModels(ASObjectList boundASModels);

    public ASModel getInternalAS();

    public void setInternalAS(ASModel as)
```

```
                              throws DOMException;

    public void addAS(ASModel as);

    public void removeAS(ASModel as)
                         throws ASException;

    public ASElementDecl getElementDecl(Element node)
                                        throws DOMException;

    public ASAttributeDecl getAttributeDecl(Attr node)
                                            throws DOMException;

    public ASEntityDecl getEntityDecl(Entity node)
                                      throws DOMException;

    public ASNotationDecl getNotationDecl(Notation node)
                                          throws DOMException;

    public void validate()
                         throws ASException;

}
```

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

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

import org.w3c.dom.DOMImplementation;

public interface DOMImplementationAS extends DOMImplementation {
    public ASWModel createASWModel(boolean isNamespaceAware,
                                   boolean container,
                                   String schemaType);

}
```

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 extends Node {
    // 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 ASException;

}
```

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/ASDOMBuilder.java:

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

import org.w3c.dom.ls.DOMInputSource;
import org.w3c.dom.ls.DOMBuilder;

public interface ASDOMBuilder extends DOMBuilder {
    public ASWModel getAbstractSchema();
    public void setAbstractSchema(ASWModel abstractSchema);

    public ASWModel parseASURI(String uri,
                               String schemaType)
                               throws ASException, Exception;

    public ASWModel parseASInputSource(DOMInputSource is,
                                       String schemaType)
                                       throws ASException, Exception;

}
```

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

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

import org.w3c.dom.ls.DOMWriter;

public interface ASDOMWriter 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,
                                       String schemaType)
                                       throws DOMException;

    public DOMWriter createDOMWriter();

    public DOMInputSource createDOMInputSource();

}
```

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

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

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

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

    public void abort();

    public boolean load(String uri);

    public boolean loadXML(String source);
    public String saveXML(Node snode)
                          throws DOMException;

}
```

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

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

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

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

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

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

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

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

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

}
```

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

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

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

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

    public DOMInputSource getInputSource();

}
```

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

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

import org.w3c.dom.events.Event;

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

    public int getTotalSize();

}
```

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

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

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

}
```

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

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

import org.w3c.dom.Node;

public interface DOMBuilderFilter {
    public static final short FILTER_INTERRUPT          = 4;
    public short startContainer(Node snode);

    public short acceptNode(Node enode);

    public int getWhatToShow();

}
```

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

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

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

public interface ParseErrorEvent extends Event {
    public DOMError getError();

}
```

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

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

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

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

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

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

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

    public boolean canSetFeature(String name,
                                 boolean state);

    public boolean getFeature(String name)
                              throws DOMException;

    public Document parseURI(String uri);

    public Document parse(DOMInputSource is)
                          throws Exception;

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

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

}
```

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

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

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

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

    public boolean canSetFeature(String name,
                                 boolean state);
    public boolean getFeature(String name)
                              throws DOMException;

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

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

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

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

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

    public String writeToString(Node wnode)
                                throws DOMException;

}
```

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

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

import org.w3c.dom.traversal.NodeFilter;

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

}
```

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

Appendix C: ECMAScript Language Binding

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

Properties of the **ASConstants** Constructor function:

ASConstants.ELEMENT_DECLARATION
The value of the constant **ASConstants.ELEMENT_DECLARATION** is **1**.

ASConstants.ATTRIBUTE_DECLARATION
The value of the constant **ASConstants.ATTRIBUTE_DECLARATION** is **2**.

ASConstants.NOTATION_DECLARATION
The value of the constant **ASConstants.NOTATION_DECLARATION** is **3**.

ASConstants.ENTITY_DECLARATION
The value of the constant **ASConstants.ENTITY_DECLARATION** is **4**.

ASConstants.CONTENT_MODEL
The value of the constant **ASConstants.CONTENT_MODEL** is **5**.

ASConstants.SCHEMA_MODEL
The value of the constant **ASConstants.SCHEMA_MODEL** is **6**.

ASConstants.INTERNAL_SUBSET
The value of the constant **ASConstants.INTERNAL_SUBSET** is **30**.

ASConstants.EXTERNAL_SUBSET
The value of the constant **ASConstants.EXTERNAL_SUBSET** is **31**.

ASConstants.NOT_USED
The value of the constant **ASConstants.NOT_USED** is **32**.

ASConstants.INTERNAL_ENTITY
The value of the constant **ASConstants.INTERNAL_ENTITY** is **33**.

ASConstants.EXTERNAL_ENTITY
The value of the constant **ASConstants.EXTERNAL_ENTITY** is **34**.

ASConstants.EMPTY_CONTENTTYPE
The value of the constant **ASConstants.EMPTY_CONTENTTYPE** is **40**.

ASConstants.SIMPLE_CONTENTTYPE
The value of the constant **ASConstants.SIMPLE_CONTENTTYPE** is **41**.

ASConstants.ELEMENT_CONTENTTYPE
The value of the constant **ASConstants.ELEMENT_CONTENTTYPE** is **42**.

ASConstants.MIXED_CONTENTTYPE
The value of the constant **ASConstants.MIXED_CONTENTTYPE** is **43**.

ASConstants.ANY_CONTENTTYPE
The value of the constant **ASConstants.ANY_CONTENTTYPE** is **44**.

ASConstants.SEQUENCE_CM
The value of the constant **ASConstants.SEQUENCE_CM** is **50**.

ASConstants.CHOICE_CM
The value of the constant **ASConstants.CHOICE_CM** is **51**.

ASConstants.ALL_CM
The value of the constant **ASConstants.ALL_CM** is **52**.

ASConstants.UNDEFINED_CM
The value of the constant **ASConstants.UNDEFINED_CM** is **53**.
ASConstants.ATOMIC_CM
The value of the constant **ASConstants.ATOMIC_CM** is **54**.
ASConstants.NONE_VC
The value of the constant **ASConstants.NONE_VC** is **0**.
ASConstants.DEFAULT_VC
The value of the constant **ASConstants.DEFAULT_VC** is **60**.
ASConstants.FIXED_VC
The value of the constant **ASConstants.FIXED_VC** is **61**.
ASConstants.REQUIRED_VC
The value of the constant **ASConstants.REQUIRED_VC** is **62**.
ASConstants.UNBOUNDED
The value of the constant **ASConstants.UNBOUNDED** is **MAX_VALUE**.

Objects that implement the **ASConstants** interface:

Objects that implement the **ASObject** interface:

Properties of objects that implement the **ASObject** interface:
objectType
This read-only property is a **Number**.
ownerModel
This read-only property is an object that implements the **ASModel** interface.
rawname
This read-only property is a **String**.
name
This read-only property is a **String**.
namespace
This read-only property is a **String**.

Functions of objects that implement the **ASObject** interface:
cloneASObject(deep)
This function returns an object that implements the **ASObject** interface.
The **deep** parameter is a **Boolean**.
This function can raise an object that implements the **ASException** interface.

Properties of the **ASDataType** Constructor function:
ASDataType.STRING_DATATYPE
The value of the constant **ASDataType.STRING_DATATYPE** is **1**.
ASDataType.NOTATION_DATATYPE
The value of the constant **ASDataType.NOTATION_DATATYPE** is **10**.
ASDataType.ID_DATATYPE
The value of the constant **ASDataType.ID_DATATYPE** is **11**.
ASDataType.IDREF_DATATYPE
The value of the constant **ASDataType.IDREF_DATATYPE** is **12**.
ASDataType.IDREFS_DATATYPE
The value of the constant **ASDataType.IDREFS_DATATYPE** is **13**.
ASDataType.ENTITY_DATATYPE
The value of the constant **ASDataType.ENTITY_DATATYPE** is **14**.

ASDataType.ENTITIES_DATATYPE
The value of the constant **ASDataType.ENTITIES_DATATYPE** is **15**.
ASDataType.NMTOKEN_DATATYPE
The value of the constant **ASDataType.NMTOKEN_DATATYPE** is **16**.
ASDataType.NMTOKENS_DATATYPE
The value of the constant **ASDataType.NMTOKENS_DATATYPE** is **17**.
ASDataType.BOOLEAN_DATATYPE
The value of the constant **ASDataType.BOOLEAN_DATATYPE** is **100**.
ASDataType.FLOAT_DATATYPE
The value of the constant **ASDataType.FLOAT_DATATYPE** is **101**.
ASDataType.DOUBLE_DATATYPE
The value of the constant **ASDataType.DOUBLE_DATATYPE** is **102**.
ASDataType.DECIMAL_DATATYPE
The value of the constant **ASDataType.DECIMAL_DATATYPE** is **103**.
ASDataType.HEXBINARY_DATATYPE
The value of the constant **ASDataType.HEXBINARY_DATATYPE** is **104**.
ASDataType.BASE64BINARY_DATATYPE
The value of the constant **ASDataType.BASE64BINARY_DATATYPE** is **105**.
ASDataType.ANYURI_DATATYPE
The value of the constant **ASDataType.ANYURI_DATATYPE** is **106**.
ASDataType.QNAME_DATATYPE
The value of the constant **ASDataType.QNAME_DATATYPE** is **107**.
ASDataType.DURATION_DATATYPE
The value of the constant **ASDataType.DURATION_DATATYPE** is **108**.
ASDataType.DATETIME_DATATYPE
The value of the constant **ASDataType.DATETIME_DATATYPE** is **109**.
ASDataType.DATE_DATATYPE
The value of the constant **ASDataType.DATE_DATATYPE** is **110**.
ASDataType.TIME_DATATYPE
The value of the constant **ASDataType.TIME_DATATYPE** is **111**.
ASDataType.GYEARMONTH_DATATYPE
The value of the constant **ASDataType.GYEARMONTH_DATATYPE** is **112**.
ASDataType.GYEAR_DATATYPE
The value of the constant **ASDataType.GYEAR_DATATYPE** is **113**.
ASDataType.GMONTHDAY_DATATYPE
The value of the constant **ASDataType.GMONTHDAY_DATATYPE** is **114**.
ASDataType.GDAY_DATATYPE
The value of the constant **ASDataType.GDAY_DATATYPE** is **115**.
ASDataType.GMONTH_DATATYPE
The value of the constant **ASDataType.GMONTH_DATATYPE** is **116**.
ASDataType.INTEGER
The value of the constant **ASDataType.INTEGER** is **117**.
ASDataType.NAME_DATATYPE
The value of the constant **ASDataType.NAME_DATATYPE** is **200**.
ASDataType.NCNAME_DATATYPE
The value of the constant **ASDataType.NCNAME_DATATYPE** is **201**.

ASDataType.NORMALIZEDSTRING_DATATYPE
The value of the constant **ASDataType.NORMALIZEDSTRING_DATATYPE** is **202**.
ASDataType.TOKEN_DATATYPE
The value of the constant **ASDataType.TOKEN_DATATYPE** is **203**.
ASDataType.LANGUAGE_DATATYPE
The value of the constant **ASDataType.LANGUAGE_DATATYPE** is **204**.
ASDataType.NONPOSITIVEINTEGER_DATATYPE
The value of the constant **ASDataType.NONPOSITIVEINTEGER_DATATYPE** is **205**.
ASDataType.NEGATIVEINTEGER_DATATYPE
The value of the constant **ASDataType.NEGATIVEINTEGER_DATATYPE** is **206**.
ASDataType.LONG_DATATYPE
The value of the constant **ASDataType.LONG_DATATYPE** is **207**.
ASDataType.INT_DATATYPE
The value of the constant **ASDataType.INT_DATATYPE** is **208**.
ASDataType.SHORT_DATATYPE
The value of the constant **ASDataType.SHORT_DATATYPE** is **209**.
ASDataType.BYTE_DATATYPE
The value of the constant **ASDataType.BYTE_DATATYPE** is **210**.
ASDataType.NONNEGATIVEINTEGER_DATATYPE
The value of the constant **ASDataType.NONNEGATIVEINTEGER_DATATYPE** is **211**.
ASDataType.UNSIGNEDLONG_DATATYPE
The value of the constant **ASDataType.UNSIGNEDLONG_DATATYPE** is **212**.
ASDataType.UNSIGNEDINT_DATATYPE
The value of the constant **ASDataType.UNSIGNEDINT_DATATYPE** is **213**.
ASDataType.UNSIGNEDSHORT_DATATYPE
The value of the constant **ASDataType.UNSIGNEDSHORT_DATATYPE** is **214**.
ASDataType.UNSIGNEDBYTE_DATATYPE
The value of the constant **ASDataType.UNSIGNEDBYTE_DATATYPE** is **215**.
ASDataType.POSITIVEINTEGER_DATATYPE
The value of the constant **ASDataType.POSITIVEINTEGER_DATATYPE** is **216**.
ASDataType.ANYSIMPLETYPE_DATATYPE
The value of the constant **ASDataType.ANYSIMPLETYPE_DATATYPE** is **216**.
ASDataType.ANYTYPE_DATATYPE
The value of the constant **ASDataType.ANYTYPE_DATATYPE** is **216**.

Objects that implement the **ASDataType** interface:
Properties of objects that implement the **ASDataType** interface:
dataType
This read-only property is a **Number**.

Objects that implement the **ASObjectList** interface:
Properties of objects that implement the **ASObjectList** interface:
length
This read-only property is a **Number**.
Functions of objects that implement the **ASObjectList** interface:
item(index)
This function returns an object that implements the **ASObject** interface.
The **index** parameter is a **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** function with that index.

Objects that implement the **ASNamedObjectMap** interface:

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

length

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

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

item(index)

This function returns an object that implements the **ASObject** interface.
The **index** parameter is a **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** function with that index.

getNamedItem(name)

This function returns an object that implements the **ASObject** interface.
The **name** parameter is a **String**.

getNamedItemNS(namespaceURI, localName)

This function returns an object that implements the **ASObject** interface.
The **namespaceURI** parameter is a **String**.
The **localName** parameter is a **String**.

Objects that implement the **ASModel** interface:

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

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

namespaceAware

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

usage

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

location

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

hint

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

container

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

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

getComponents(objectType)

This function returns an object that implements the **ASNamedObjectMap** interface.
The **objectType** parameter is a **Number**.

getElementDecl(name, targetNamespace)

This function returns an object that implements the **ASElementDecl** interface.
The **name** parameter is a **String**.
The **targetNamespace** parameter is a **String**.

getAttributeDecl(name, targetNamespace)

This function returns an object that implements the **ASAttributeDecl** interface.
The **name** parameter is a **String**.

The **targetNamespace** parameter is a **String**.

getEntityDecl(name)

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

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

getNotationDecl(name, targetNamespace)

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

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

The **targetNamespace** parameter is a **String**.

Objects that implement the **ASContentModel** interface:

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

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

contentModelType

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

minOccurs

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

maxOccurs

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

subModels

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

Objects that implement the **ASElementDecl** interface:

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

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

type

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

contentType

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

strictMixedContent

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

contentModel

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

isPCDataOnly

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

attributeDecls

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

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

getAttributeDecl(name, targetNamespace)

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

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

The **targetNamespace** parameter is a **String**.

Objects that implement the **ASAttributeDecl** interface:

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

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

type
This read-only property is an object that implements the **ASDataType** interface.
enumAttr
This read-only property is a **String**.
ownerElementDeclarations
This read-only property is an object that implements the **ASObjectList** interface.
defaultType
This read-only property is a **Number**.
value
This read-only property is a **String**.

Objects that implement the **ASEntityDecl** interface:
Objects that implement the **ASEntityDecl** interface have all properties and functions of the **ASObject** interface as well as the properties and functions defined below.
Properties of objects that implement the **ASEntityDecl** interface:
entityType
This read-only property is a **Number**.
entityValue
This read-only property is a **String**.
systemId
This read-only property is a **String**.
publicId
This read-only property is a **String**.

Objects that implement the **ASNotationDecl** interface:
Objects that implement the **ASNotationDecl** interface have all properties and functions of the **ASObject** interface as well as the properties and functions defined below.
Properties of objects that implement the **ASNotationDecl** interface:
systemId
This read-only property is a **String**.
publicId
This read-only property is a **String**.

Objects that implement the **ASWModel** interface:
Objects that implement the **ASWModel** interface have all properties and functions of the **ASModel** interface as well as the properties and functions defined below.
Functions of objects that implement the **ASWModel** interface:
setLocation(location)
This function has no return value.
The **location** parameter is a **String**.
setHint(hint)
This function has no return value.
The **hint** parameter is a **String**.
addComponent(declaration)
This function has no return value.
The **declaration** parameter is an object that implements the **ASObject** interface.
removeComponent(declaration)
This function has no return value.
The **declaration** parameter is an object that implements the **ASObject** interface.

addASModel(declaration)
This function has no return value.
The **declaration** parameter is an object that implements the **ASModel** interface.
removeASModel(declaration)
This function has no return value.
The **declaration** parameter is an object that implements the **ASModel** interface.
getASModels()
This function returns an object that implements the **ASObjectList** interface.
importASObject(asobject)
This function returns an object that implements the **ASObject** interface.
The **asobject** parameter is an object that implements the **ASObject** interface.
insertASObject(asobject)
This function has no return value.
The **asobject** parameter is an object that implements the **ASObject** interface.
validate()
This function returns a **Boolean**.
createASWElementDecl(namespaceURI, name)
This function returns an object that implements the **ASWElementDecl** interface.
The **namespaceURI** parameter is a **String**.
The **name** parameter is a **String**.
This function can raise an object that implements the **ASException** interface.
createASWAttributeDecl(namespaceURI, name)
This function returns an object that implements the **ASWAttributeDecl** interface.
The **namespaceURI** parameter is a **String**.
The **name** parameter is a **String**.
This function can raise an object that implements the **ASException** interface.
createASWNotationDecl(namespaceURI, name, systemId, publicId)
This function returns an object that implements the **ASWNotationDecl** interface.
The **namespaceURI** parameter is a **String**.
The **name** parameter is a **String**.
The **systemId** parameter is a **String**.
The **publicId** parameter is a **String**.
This function can raise an object that implements the **ASException** interface.
createASWEntityDecl(name)
This function returns an object that implements the **ASWEntityDecl** interface.
The **name** parameter is a **String**.
This function can raise an object that implements the **ASException** interface.
createASWContentModel(name, namespaceURI, minOccurs, maxOccurs, operator)
This function returns an object that implements the **ASWContentModel** interface.
The **name** parameter is a **String**.
The **namespaceURI** parameter is a **String**.
The **minOccurs** parameter is a **Number**.
The **maxOccurs** parameter is a **Number**.
The **operator** parameter is a **Number**.
This function can raise an object that implements the **ASException** interface.

Objects that implement the **ASWNamedObjectMap** interface:

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

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

removeNamedItem(name)

This function returns an object that implements the **ASObject** interface.
The **name** parameter is a **String**.
This function can raise an object that implements the **ASException** interface.

setNamedItem(newASObject)

This function returns an object that implements the **ASObject** interface.
The **newASObject** parameter is an object that implements the **ASObject** interface.
This function can raise an object that implements the **ASException** interface or the **ASException** interface.

setNamedItemNS(arg)

This function returns an object that implements the **ASObject** interface.
The **arg** parameter is an object that implements the **ASObject** interface.
This function can raise an object that implements the **ASException** interface.

removeNamedItemNS(namespaceURI, localName)

This function returns an object that implements the **ASObject** interface.
The **namespaceURI** parameter is a **String**.
The **localName** parameter is a **String**.
This function can raise an object that implements the **DOMException** interface.

Objects that implement the **ASWElementDecl** interface:

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

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

setRawname(rawname)

This function has no return value.
The **rawname** parameter is a **String**.

setName(name)

This function has no return value.
The **name** parameter is a **String**.

setNamespace(namespaceURI)

This function has no return value.
The **namespaceURI** parameter is a **String**.

setStrictMixedContent(mixedContent)

This function has no return value.
The **mixedContent** parameter is a **Boolean**.

setType(type)

This function has no return value.
The **type** parameter is an object that implements the **ASDataType** interface.

setContentType(contentType)

This function has no return value.
The **contentType** parameter is a **Number**.

setContentModel(contentModel)

This function has no return value.

The **contentModel** parameter is an object that implements the **ASWContentModel** interface.

addAttributeDecl(attributeDecl)

This function has no return value.

The **attributeDecl** parameter is an object that implements the **ASWAttributeDecl** interface.

removeAttributeDecl(attributeDecl)

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

The **attributeDecl** parameter is an object that implements the **ASWAttributeDecl** interface.

Objects that implement the **ASWContentModel** interface:

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

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

setName(name)

This function has no return value.

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

setNamespaceURI(namespaceURI)

This function has no return value.

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

setContentModelType(operator)

This function has no return value.

The **operator** parameter is a **Number**.

setMinOccurs(minOccurs)

This function has no return value.

The **minOccurs** parameter is a **Number**.

setMaxOccurs(maxOccurs)

This function has no return value.

The **maxOccurs** parameter is a **Number**.

removeSubModel(oldObject)

This function has no return value.

The **oldObject** parameter is an object that implements the **ASObject** interface.

insertBeforeSubModel(newObject, refObject)

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

The **newObject** parameter is an object that implements the **ASObject** interface.

The **refObject** parameter is an object that implements the **ASObject** interface.

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

appendSubModel(newObject)

This function returns a **Number**.

The **newObject** parameter is an object that implements the **ASObject** interface.

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

Objects that implement the **ASWAttributeDecl** interface:

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

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

setRawname(rawname)
This function has no return value.
The **rawname** parameter is a **String**.
setName(name)
This function has no return value.
The **name** parameter is a **String**.
setNamespaceURI(namespaceURI)
This function has no return value.
The **namespaceURI** parameter is a **String**.
setType(type)
This function has no return value.
The **type** parameter is an object that implements the **ASDataType** interface.
setValue(value)
This function has no return value.
The **value** parameter is a **String**.
setEnumAttr(enumeration)
This function has no return value.
The **enumeration** parameter is a **String**.
setDefaultType(constraint)
This function has no return value.
The **constraint** parameter is a **Number**.

Objects that implement the **ASWEntityDecl** interface:

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

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

setRawname(rawname)
This function has no return value.
The **rawname** parameter is a **String**.
setEntityType(type)
This function has no return value.
The **type** parameter is a **Number**.
setEntityValue(value)
This function has no return value.
The **value** parameter is a **String**.
setSystemId(systemId)
This function has no return value.
The **systemId** parameter is a **String**.
setPublicId(publicId)
This function has no return value.
The **publicId** parameter is a **String**.

Objects that implement the **ASWNotationDecl** interface:

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

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

setRawname(rawname)
This function has no return value.

The **rawname** parameter is a **String**.

setName(name)

This function has no return value.

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

setNamespaceURI(namespaceURI)

This function has no return value.

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

setSystemId(systemId)

This function has no return value.

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

setPublicId(publicId)

This function has no return value.

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

Properties of the **ASException** Constructor function:

ASException.DUPLICATE_NAME_ERR

The value of the constant **ASException.DUPLICATE_NAME_ERR** is **1**.

ASException.TYPE_ERR

The value of the constant **ASException.TYPE_ERR** is **2**.

ASException.NO_AS_AVAILABLE

The value of the constant **ASException.NO_AS_AVAILABLE** is **3**.

ASException.WRONG_MIME_TYPE_ERR

The value of the constant **ASException.WRONG_MIME_TYPE_ERR** is **4**.

ASException.INVALID_CHARACTER_ERR

The value of the constant **ASException.INVALID_CHARACTER_ERR** is **5**.

ASException.VALIDATION_ERR

The value of the constant **ASException.VALIDATION_ERR** is **6**.

ASException.ACTIVEAS_DELETION_ERR

The value of the constant **ASException.ACTIVEAS_DELETION_ERR** is **7**.

Objects that implement the **ASException** interface:

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

code

This property is a **Number**.

Objects that implement the **DocumentAS** interface:

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

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

activeASModel

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

boundASModels

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

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

getInternalAS()

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

setInternalAS(as)

This function has no return value.

The **as** parameter is an object that implements the **ASModel** interface.

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

addAS(as)

This function has no return value.

The **as** parameter is an object that implements the **ASModel** interface.

removeAS(as)

This function has no return value.

The **as** parameter is an object that implements the **ASModel** interface.

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

getElementDecl(node)

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

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

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

getAttributeDecl(node)

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

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

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

getEntityDecl(node)

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

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

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

getNotationDecl(node)

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

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

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

validate()

This function has no return value.

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

Objects that implement the **DOMImplementationAS** interface:

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

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

createASWModel(isNamespaceAware, container, schemaType)

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

The **isNamespaceAware** parameter is a **Boolean**.

The **container** parameter is a **Boolean**.

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

Objects that implement the **DocumentEditAS** interface:

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

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

continuousValidityChecking

This property is a **Boolean**.

Properties of the **NodeEditAS** Constructor function:

NodeEditAS.WF_CHECK

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

NodeEditAS.NS_WF_CHECK
The value of the constant **NodeEditAS.NS_WF_CHECK** is **2**.
NodeEditAS.PARTIAL_VALIDITY_CHECK
The value of the constant **NodeEditAS.PARTIAL_VALIDITY_CHECK** is **3**.
NodeEditAS.STRICT_VALIDITY_CHECK
The value of the constant **NodeEditAS.STRICT_VALIDITY_CHECK** is **4**.
Objects that implement the **NodeEditAS** interface:
Objects that implement the **NodeEditAS** interface have all properties and functions of the **Node** interface as well as the properties and functions defined below.
Functions of objects that implement the **NodeEditAS** interface:
canInsertBefore(newChild, refChild)
This function returns a **Boolean**.
The **newChild** parameter is an object that implements the **Node** interface.
The **refChild** parameter is an object that implements the **Node** interface.
canRemoveChild(oldChild)
This function returns a **Boolean**.
The **oldChild** parameter is an object that implements the **Node** interface.
canReplaceChild(newChild, oldChild)
This function returns a **Boolean**.
The **newChild** parameter is an object that implements the **Node** interface.
The **oldChild** parameter is an object that implements the **Node** interface.
canAppendChild(newChild)
This function returns a **Boolean**.
The **newChild** parameter is an object that implements the **Node** interface.
isNodeValid(deep, wFValidityCheckLevel)
This function returns a **Boolean**.
The **deep** parameter is a **Boolean**.
The **wFValidityCheckLevel** parameter is a **Number**.
This function can raise an object that implements the **ASException** interface.
Objects that implement the **ElementEditAS** interface:
Objects that implement the **ElementEditAS** interface have all properties and functions of the **NodeEditAS** interface as well as the properties and functions defined below.
Properties of objects that implement the **ElementEditAS** interface:
definedElementTypes
This read-only property is an object that implements the **NodeList** interface.
Functions of objects that implement the **ElementEditAS** interface:
contentType()
This function returns a **Number**.
canSetAttribute(attrname, attrval)
This function returns a **Boolean**.
The **attrname** parameter is a **String**.
The **attrval** parameter is a **String**.
canSetAttributeNode(attrNode)
This function returns a **Boolean**.
The **attrNode** parameter is an object that implements the **Attr** interface.

canSetAttributeNS(name, attrval, namespaceURI)
This function returns a **Boolean**.
The **name** parameter is a **String**.
The **attrval** parameter is a **String**.
The **namespaceURI** parameter is a **String**.
canRemoveAttribute(attrname)
This function returns a **Boolean**.
The **attrname** parameter is a **String**.
canRemoveAttributeNS(attrname, namespaceURI)
This function returns a **Boolean**.
The **attrname** parameter is a **String**.
The **namespaceURI** parameter is a **String**.
canRemoveAttributeNode(attrNode)
This function returns a **Boolean**.
The **attrNode** parameter is an object that implements the **Node** interface.
getChildElements()
This function returns an object that implements the **NodeList** interface.
getParentElements()
This function returns an object that implements the **NodeList** interface.
getAttributeList()
This function returns an object that implements the **NodeList** interface.
isElementDefined(elemTypeName)
This function returns a **Boolean**.
The **elemTypeName** parameter is a **String**.
isElementDefinedNS(elemTypeName, namespaceURI, name)
This function returns a **Boolean**.
The **elemTypeName** parameter is a **String**.
The **namespaceURI** parameter is a **String**.
The **name** parameter is a **String**.

Objects that implement the **CharacterDataEditAS** interface:
Objects that implement the **CharacterDataEditAS** interface have all properties and functions of the **NodeEditAS** interface as well as the properties and functions defined below.
Properties of objects that implement the **CharacterDataEditAS** interface:
isWhitespaceOnly
This read-only property is a **Boolean**.
Functions of objects that implement the **CharacterDataEditAS** interface:
canSetData(offset, count)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.
canAppendData(arg)
This function returns a **Boolean**.
The **arg** parameter is a **String**.
canReplaceData(offset, count, arg)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.

The **count** parameter is a **Number**.
The **arg** parameter is a **String**.
canInsertData(offset, arg)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **arg** parameter is a **String**.
canDeleteData(offset, count)
This function returns a **Boolean**.
The **offset** parameter is a **Number**.
The **count** parameter is a **Number**.
Objects that implement the **ASDOMBuilder** interface:
Objects that implement the **ASDOMBuilder** interface have all properties and functions of the **DOMBuilder** interface as well as the properties and functions defined below.
Properties of objects that implement the **ASDOMBuilder** interface:
abstractSchema
This property is an object that implements the **ASWModel** interface.
Functions of objects that implement the **ASDOMBuilder** interface:
parseASURI(uri, schemaType)
This function returns an object that implements the **ASWModel** interface.
The **uri** parameter is a **String**.
The **schemaType** parameter is a **String**.
This function can raise an object that implements the **ASException** interface or the **DOMSystemException** interface.
parseASInputSource(is, schemaType)
This function returns an object that implements the **ASWModel** interface.
The **is** parameter is an object that implements the **DOMInputSource** interface.
The **schemaType** parameter is a **String**.
This function can raise an object that implements the **ASException** interface or the **DOMSystemException** interface.
Objects that implement the **ASDOMWriter** interface:
Objects that implement the **ASDOMWriter** interface have all properties and functions of the **DOMWriter** interface as well as the properties and functions defined below.
Functions of objects that implement the **ASDOMWriter** interface:
writeASModel(destination, model)
This function has no return value.
The **destination** parameter is an object that implements the **Object** interface.
The **model** parameter is an object that implements the **ASModel** interface.
This function can raise an object that implements the **DOMSystemException** interface.

Properties of the **DOMImplementationLS** Constructor function:
DOMImplementationLS.MODE_SYNCHRONOUS
The value of the constant **DOMImplementationLS.MODE_SYNCHRONOUS** is **1**.
DOMImplementationLS.MODE_ASYNCHRONOUS
The value of the constant **DOMImplementationLS.MODE_ASYNCHRONOUS** is **2**.
Objects that implement the **DOMImplementationLS** interface:

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

createDOMBuilder(mode, schemaType)

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

createDOMWriter()

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

createDOMInputSource()

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

Objects that implement the **DocumentLS** interface:

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

async

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

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

abort()

This function has no return value.

load(uri)

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

loadXML(source)

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

saveXML(snode)

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

Objects that implement the **DOMInputSource** interface:

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

byteStream

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

characterStream

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

stringData

This property is a **String**.

encoding

This property is a **String**.

publicId

This property is a **String**.

systemId

This property is a **String**.

baseURI

This property is a **String**.

Objects that implement the **LSLoadEvent** interface:

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

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

newDocument

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

inputSource

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

Objects that implement the **LSProgressEvent** interface:

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

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

inputSource

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

position

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

totalSize

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

Objects that implement the **DOMEntityResolver** interface:

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

resolveEntity(publicId, systemId, baseURI)

This function returns an object that implements the **DOMInputSource** interface.
The **publicId** parameter is a **String**.
The **systemId** parameter is a **String**.
The **baseURI** parameter is a **String**.
This function can raise an object that implements the **DOMSystemException** interface.

Properties of the **DOMBuilderFilter** Constructor function:

DOMBuilderFilter.FILTER_INTERRUPT

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

Objects that implement the **DOMBuilderFilter** interface:

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

whatToShow

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

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

startContainer(snode)

This function returns a **Number**.
The **snode** parameter is an object that implements the **Node** interface.

acceptNode(enode)

This function returns a **Number**.
The **enode** parameter is an object that implements the **Node** interface.

Objects that implement the **ParseErrorEvent** interface:

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

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

error

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

Properties of the **DOMBuilder** Constructor function:

- **DOMBuilder.ACTION_REPLACE**
  - The value of the constant **DOMBuilder.ACTION_REPLACE** is **1**.
- **DOMBuilder.ACTION_APPEND_AS_CHILDREN**
  - The value of the constant **DOMBuilder.ACTION_APPEND_AS_CHILDREN** is **2**.
- **DOMBuilder.ACTION_INSERT_AFTER**
  - The value of the constant **DOMBuilder.ACTION_INSERT_AFTER** is **3**.
- **DOMBuilder.ACTION_INSERT_BEFORE**
  - The value of the constant **DOMBuilder.ACTION_INSERT_BEFORE** is **4**.

Objects that implement the **DOMBuilder** interface:

- Properties of objects that implement the **DOMBuilder** interface:
  - **entityResolver**
    - This property is an object that implements the **DOMEntityResolver** interface.
  - **errorHandler**
    - This property is an object that implements the **DOMErrorHandler** interface.
  - **filter**
    - This property is an object that implements the **DOMBuilderFilter** interface.
- Functions of objects that implement the **DOMBuilder** interface:
  - **setFeature(name, state)**
    - This function has no return value.
    - The **name** parameter is a **String**.
    - The **state** parameter is a **Boolean**.
    - This function can raise an object that implements the **DOMException** interface.
  - **canSetFeature(name, state)**
    - This function returns a **Boolean**.
    - The **name** parameter is a **String**.
    - The **state** parameter is a **Boolean**.
  - **getFeature(name)**
    - This function returns a **Boolean**.
    - The **name** parameter is a **String**.
    - This function can raise an object that implements the **DOMException** interface.
  - **parseURI(uri)**
    - This function returns an object that implements the **Document** interface.
    - The **uri** parameter is a **String**.
  - **parse(is)**
    - This function returns an object that implements the **Document** interface.
    - The **is** parameter is an object that implements the **DOMInputSource** interface.
    - This function can raise an object that implements the **DOMSystemException** interface.
  - **parseWithContext(is, cnode, action)**
    - This function has no return value.
    - The **is** parameter is an object that implements the **DOMInputSource** interface.
    - The **cnode** parameter is an object that implements the **Node** interface.
    - The **action** parameter is a **Number**.
    - This function can raise an object that implements the **DOMException** interface.

Objects that implement the **DOMWriter** interface:

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

encoding

This property is a **String**.

newLine

This property is a **String**.

filter

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

errorHandler

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

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

setFeature(name, state)

This function has no return value.
The **name** parameter is a **String**.
The **state** parameter is a **Boolean**.
This function can raise an object that implements the **DOMException** interface.

canSetFeature(name, state)

This function returns a **Boolean**.
The **name** parameter is a **String**.
The **state** parameter is a **Boolean**.

getFeature(name)

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

writeNode(destination, wnode)

This function returns a **Boolean**.
The **destination** parameter is an object that implements the **Object** interface.
The **wnode** parameter is an object that implements the **Node** interface.
This function can raise an object that implements the **DOMSystemException** interface.

writeToString(wnode)

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

Objects that implement the **DOMWriterFilter** interface:

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

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

whatToShow

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

Appendix D: Acknowledgements

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

Andrew Watson (Object Management Group), Andy Heninger (IBM), Angel Diaz (IBM), Arnaud Le Hors (W3C and IBM), Ashok Malhotra (IBM and Microsoft), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), Chris Lovett (Microsoft), Chris Wilson (Microsoft), David Brownell (Sun), David Ezell (Hewlett Packard Company), David Singer (IBM), Dimitris Dimitriadis (Improve AB), Don Park (invited), Elena Litani (IBM), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Jeroen van Rotterdam (X-Hive Corporation), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape/AOL), Jon Ferraiolo (Adobe), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Lauren Wood (SoftQuad Software Inc., *former chair*), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mary Brady (NIST), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégaret (W3C, *W3C team contact and 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 1.0].

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

document order
: There is an ordering, *document order*, defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities. Thus, the *document element* [p.163] node will be the first node. Element nodes occur before their children. Thus, document order orders element nodes in order of the occurrence of their start-tag in the XML (after expansion of entities). The attribute nodes of an element occur after the element and before its children. The relative order of attribute nodes is implementation-dependent.

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 1.0].

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 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.164] (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*.

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 1.0]

References

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

F.1: Normative references

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

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

[DOM Level 3 Core]
Document Object Model Level 3 Core Specification, A. Le Hors, et al., Editors. World Wide Web Consortium, January 2002. This version of the Document Object Model Level 3 Core Specification is http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20020114. The latest version of DOM Level 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core.

[ECMAScript]
ECMAScript Language Specification, Third Edition. European Computer Manufacturers Association, December 1999. This version of the ECMAScript Language is available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM.

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

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

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

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

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

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

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

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

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

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

[XML Schema Part 0]
XML Schema Part 0: Primer, D. Fallside, Editor. World Wide Web Consortium, 2 May 2001. This version of the XML Part 0 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-0-20010502. The latest version of XML Schema Part 0 is available at http://www.w3.org/TR/xmlschema-0.

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

[XML Schema Part 2]
XML Schema Part 2: Datatypes, P. Byron and Ashok Malhotra, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Part 2 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2.

F.2: Informative references

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

[COM]
The Microsoft Component Object Model, Microsoft Corporation. Available at http://www.microsoft.com/com.

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

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

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

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

Index