



# XML Schema Part 1: Structures

## Second Edition

**W3C Recommendation 28 October 2004**

This version:

<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>

Latest version:

<http://www.w3.org/TR/xmlschema-1/>

Previous version:

<http://www.w3.org/TR/2004/PER-xmlschema-1-20040318/>

Authors and Contributors:

Henry S. Thompson (University of Edinburgh) <[ht@cogsci.ed.ac.uk](mailto:ht@cogsci.ed.ac.uk)>

David Beech (Oracle Corporation) <[David.Beech@oracle.com](mailto:David.Beech@oracle.com)>

Murray Maloney (for Commerce One) <[murray@muzmo.com](mailto:murray@muzmo.com)>

Noah Mendelsohn (Lotus Development Corporation) <[Noah\\_Mendelsohn@lotus.com](mailto:Noah_Mendelsohn@lotus.com)>

Copyright © 2004 W3C<sup>®</sup> (MIT, INRIA, Keio), All Rights Reserved.  
W3C liability, trademark, document use, and software licensing rules apply.

### Abstract

*XML Schema: Structures* specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs). This specification depends on *XML Schema Part 2: Datatypes*.

---

## Status of this document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](http://www.w3.org/TR/) at <http://www.w3.org/TR/>.*

This is a [W3C Recommendation](#), which forms part of the Second Edition of XML Schema. This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced by the [W3C XML Schema Working Group](#) as part of the W3C [XML Activity](#). The goals of the XML Schema language are discussed in the [XML Schema Requirements](#) document. The authors of this document are the members of the XML Schema Working Group. Different parts of this specification have different editors.

This document was produced under the [24 January 2002 Current Patent Practice \(CPP\)](#) as amended by the [W3C Patent Policy Transition Procedure](#). The Working Group maintains a [public list of patent disclosures](#) relevant to this document; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

The English version of this specification is the only normative version. Information about translations of this document is available at <http://www.w3.org/2001/05/xmlschema-translations>.

This second edition is *not* a new version, it merely incorporates the changes dictated by the corrections to errors found in the [first edition](#) as agreed by the XML Schema Working Group, as a convenience to readers. A separate list of all such corrections is available at <http://www.w3.org/2001/05/xmlschema-errata>.

The errata list for this second edition is available at <http://www.w3.org/2004/03/xmlschema-errata>.

Please report errors in this document to [www-xml-schema-comments@w3.org](mailto:www-xml-schema-comments@w3.org) ([archive](#)).



David Beech has retired since the publication of the first edition, and can be reached at [davidbeech@earthlink.net](mailto:davidbeech@earthlink.net).

Murray Maloney is no longer affiliated with Commerce One; his contact details are unchanged.

Noah Mendelsohn's affiliation has changed since the publication of the first edition. He is now at IBM, and can be contacted at [noah\\_mendelsohn@us.ibm.com](mailto:noah_mendelsohn@us.ibm.com)

---

# Table of Contents

<b>1. Introduction</b> .....	<b>1</b>
1.1. Purpose .....	1
1.2. Dependencies on Other Specifications .....	2
1.3. Documentation Conventions and Terminology .....	2
<b>2. Conceptual Framework</b> .....	<b>3</b>
2.1. Overview of XML Schema .....	3
2.2. XML Schema Abstract Data Model .....	4
2.2.1. Type Definition Components .....	5
2.2.1.1. Type Definition Hierarchy .....	5
2.2.1.2. Simple Type Definition .....	6
2.2.1.3. Complex Type Definition .....	6
2.2.2. Declaration Components .....	7
2.2.2.1. Element Declaration .....	7
2.2.2.2. Element Substitution Group .....	7
2.2.2.3. Attribute Declaration .....	7
2.2.2.4. Notation Declaration .....	8
2.2.3. Model Group Components .....	8
2.2.3.1. Model Group .....	8
2.2.3.2. Particle .....	8
2.2.3.3. Attribute Use .....	8
2.2.3.4. Wildcard .....	8
2.2.4. Identity-constraint Definition Components .....	9
2.2.5. Group Definition Components .....	9
2.2.5.1. Model Group Definition .....	9
2.2.5.2. Attribute Group Definition .....	9
2.2.6. Annotation Components .....	9
2.3. Constraints and Validation Rules .....	9
2.4. Conformance .....	10
2.5. Names and Symbol Spaces .....	11
2.6. Schema-Related Markup in Documents Being Validated .....	11
2.6.1. xsi:type .....	12
2.6.2. xsi:nil .....	12
2.6.3. xsi:schemaLocation, xsi:noNamespaceSchemaLocation .....	12
2.7. Representation of Schemas on the World Wide Web .....	12
<b>3. Schema Component Details</b> .....	<b>13</b>
3.1. Introduction .....	13
3.1.1. Components and Properties .....	13
3.1.2. XML Representations of Components .....	14
3.1.3. The Mapping between XML Representations and Components .....	14
3.1.4. White Space Normalization during Validation .....	15

---

3.2. Attribute Declarations .....	16
3.2.1. The Attribute Declaration Schema Component .....	16
3.2.2. XML Representation of Attribute Declaration Schema Components .....	17
3.2.3. Constraints on XML Representations of Attribute Declarations .....	18
3.2.4. Attribute Declaration Validation Rules .....	18
3.2.5. Attribute Declaration Information Set Contributions .....	19
3.2.6. Constraints on Attribute Declaration Schema Components .....	21
3.2.7. Built-in Attribute Declarations .....	22
3.3. Element Declarations .....	22
3.3.1. The Element Declaration Schema Component .....	23
3.3.2. XML Representation of Element Declaration Schema Components .....	24
3.3.3. Constraints on XML Representations of Element Declarations .....	27
3.3.4. Element Declaration Validation Rules .....	28
3.3.5. Element Declaration Information Set Contributions .....	33
3.3.6. Constraints on Element Declaration Schema Components .....	36
3.4. Complex Type Definitions .....	38
3.4.1. The Complex Type Definition Schema Component .....	39
3.4.2. XML Representation of Complex Type Definitions .....	40
3.4.3. Constraints on XML Representations of Complex Type Definitions .....	47
3.4.4. Complex Type Definition Validation Rules .....	48
3.4.5. Complex Type Definition Information Set Contributions .....	50
3.4.6. Constraints on Complex Type Definition Schema Components .....	51
3.4.7. Built-in Complex Type Definition .....	55
3.5. AttributeUses .....	55
3.5.1. The Attribute Use Schema Component .....	55
3.5.2. XML Representation of Attribute Use Components .....	56
3.5.3. Constraints on XML Representations of Attribute Uses .....	56
3.5.4. Attribute Use Validation Rules .....	56
3.5.5. Attribute Use Information Set Contributions .....	56
3.5.6. Constraints on Attribute Use Schema Components .....	56
3.6. Attribute Group Definitions .....	56
3.6.1. The Attribute Group Definition Schema Component .....	57
3.6.2. XML Representation of Attribute Group Definition Schema Components .....	57
3.6.3. Constraints on XML Representations of Attribute Group Definitions .....	57
3.6.4. Attribute Group Definition Validation Rules .....	58
3.6.5. Attribute Group Definition Information Set Contributions .....	58
3.6.6. Constraints on Attribute Group Definition Schema Components .....	58
3.7. Model Group Definitions .....	58
3.7.1. The Model Group Definition Schema Component .....	59
3.7.2. XML Representation of Model Group Definition Schema Components .....	59
3.7.3. Constraints on XML Representations of Model Group Definitions .....	60
3.7.4. Model Group Definition Validation Rules .....	60

---

---

3.7.5. Model Group Definition Information Set Contributions .....	60
3.7.6. Constraints on Model Group Definition Schema Components .....	60
3.8. Model Groups .....	60
3.8.1. The Model Group Schema Component .....	61
3.8.2. XML Representation of Model Group Schema Components .....	61
3.8.3. Constraints on XML Representations of Model Groups .....	62
3.8.4. Model Group Validation Rules .....	62
3.8.5. Model Group Information Set Contributions .....	63
3.8.6. Constraints on Model Group Schema Components .....	63
3.9. Particles .....	65
3.9.1. The Particle Schema Component .....	65
3.9.2. XML Representation of Particle Components .....	65
3.9.3. Constraints on XML Representations of Particles .....	65
3.9.4. Particle Validation Rules .....	65
3.9.5. Particle Information Set Contributions .....	68
3.9.6. Constraints on Particle Schema Components .....	68
3.10. Wildcards .....	74
3.10.1. The Wildcard Schema Component .....	74
3.10.2. XML Representation of Wildcard Schema Components .....	75
3.10.3. Constraints on XML Representations of Wildcards .....	76
3.10.4. Wildcard Validation Rules .....	76
3.10.5. Wildcard Information Set Contributions .....	77
3.10.6. Constraints on Wildcard Schema Components .....	77
3.11. Identity-constraint Definitions .....	79
3.11.1. The Identity-constraint Definition Schema Component .....	79
3.11.2. XML Representation of Identity-constraint Definition Schema Components .....	80
3.11.3. Constraints on XML Representations of Identity-constraint Definitions .....	82
3.11.4. Identity-constraint Definition Validation Rules .....	82
3.11.5. Identity-constraint Definition Information Set Contributions .....	84
3.11.6. Constraints on Identity-constraint Definition Schema Components .....	85
3.12. Notation Declarations .....	86
3.12.1. The Notation Declaration Schema Component .....	86
3.12.2. XML Representation of Notation Declaration Schema Components .....	86
3.12.3. Constraints on XML Representations of Notation Declarations .....	87
3.12.4. Notation Declaration Validation Rules .....	87
3.12.5. Notation Declaration Information Set Contributions .....	87
3.12.6. Constraints on Notation Declaration Schema Components .....	88
3.13. Annotations .....	88
3.13.1. The Annotation Schema Component .....	88
3.13.2. XML Representation of Annotation Schema Components .....	89
3.13.3. Constraints on XML Representations of Annotations .....	89
3.13.4. Annotation Validation Rules .....	89

---

3.13.5. Annotation Information Set Contributions .....	89
3.13.6. Constraints on Annotation Schema Components .....	89
3.14. Simple Type Definitions .....	89
3.14.1. (non-normative) The Simple Type Definition Schema Component .....	90
3.14.2. (non-normative) XML Representation of Simple Type Definition Schema Components .....	91
3.14.3. Constraints on XML Representations of Simple Type Definitions .....	92
3.14.4. Simple Type Definition Validation Rules .....	93
3.14.5. Simple Type Definition Information Set Contributions .....	93
3.14.6. Constraints on Simple Type Definition Schema Components .....	93
3.14.7. Built-in Simple Type Definition .....	96
3.15. Schemas as a Whole .....	96
3.15.1. The Schema Itself .....	96
3.15.2. XML Representations of Schemas .....	97
3.15.2.1. References to Schema Components .....	98
3.15.2.2. References to Schema Components from Elsewhere .....	99
3.15.3. Constraints on XML Representations of Schemas .....	99
3.15.4. Validation Rules for Schemas as a Whole .....	101
3.15.5. Schema Information Set Contributions .....	101
3.15.6. Constraints on Schemas as a Whole .....	103
<b>4. Schemas and Namespaces: Access and Composition .....</b>	<b>103</b>
4.1. Layer 1: Summary of the Schema-validity Assessment Core .....	104
4.2. Layer 2: Schema Documents, Namespaces and Composition .....	105
4.2.1. Assembling a schema for a single target namespace from multiple schema definition documents .....	105
4.2.2. Including modified component definitions .....	106
4.2.3. References to schema components across namespaces .....	110
4.3. Layer 3: Schema Document Access and Web-interoperability .....	112
4.3.1. Standards for representation of schemas and retrieval of schema documents on the Web .....	113
4.3.2. How schema definitions are located on the Web .....	113
<b>5. Schemas and Schema-validity Assessment .....</b>	<b>115</b>
5.1. Errors in Schema Construction and Structure .....	115
5.2. Assessing Schema-Validity .....	116
5.3. Missing Sub-components .....	117
5.4. Responsibilities of Schema-aware Processors .....	118
 <b>Appendices</b>	
<b>A. Schema for Schemas (normative) .....</b>	<b>118</b>
<b>B. References (normative) .....</b>	<b>118</b>

---

<b>C. Outcome Tabulations (normative)</b> .....	<b>119</b>
C.1. Validation Rules .....	119
C.2. Contributions to the post-schema-validation infoset .....	119
C.3. Schema Representation Constraints .....	119
C.4. Schema Component Constraints .....	119
<b>D. Required Information Set Items and Properties (normative)</b> .....	<b>119</b>
<b>E. Schema Components Diagram (non-normative)</b> .....	<b>120</b>
<b>F. Glossary (non-normative)</b> .....	<b>120</b>
<b>G. DTD for Schemas (non-normative)</b> .....	<b>120</b>
<b>H. Analysis of the Unique Particle Attribution Constraint (non-normative)</b> .....	<b>121</b>
<b>I. References (non-normative)</b> .....	<b>122</b>
<b>J. Acknowledgements (non-normative)</b> .....	<b>122</b>

*This page is intentionally left blank.*



# 1. Introduction

This document sets out the structural part (*XML Schema: Structures*) of the XML Schema definition language.

Chapter 2 presents a [§ 2 – Conceptual Framework](#) on page 3 for XML Schemas, including an introduction to the nature of XML Schemas and an introduction to the XML Schema abstract data model, along with other terminology used throughout this document.

Chapter 3, [§ 3 – Schema Component Details](#) on page 13, specifies the precise semantics of each component of the abstract model, the representation of each component in XML, with reference to a DTD and XML Schema for an XML Schema document type, along with a detailed mapping between the elements and attribute vocabulary of this representation and the components and properties of the abstract model.

Chapter 4 presents [§ 4 – Schemas and Namespaces: Access and Composition](#) on page 103, including the connection between documents and schemas, the import, inclusion and redefinition of declarations and definitions and the foundations of schema-validity assessment.

Chapter 5 discusses [§ 5 – Schemas and Schema-validity Assessment](#) on page 115, including the overall approach to schema-validity assessment of documents, and responsibilities of schema-aware processors.

The normative appendices include a [Appendix A – Schema for Schemas \(normative\)](#) on page 118 for the XML representation of schemas and [Appendix B – References \(normative\)](#) on page 118.

The non-normative appendices include the [Appendix G – DTD for Schemas \(non-normative\)](#) on page 120 and a [Appendix F – Glossary \(non-normative\)](#) on page 120.

This document is primarily intended as a language definition reference. As such, although it contains a few examples, it is *not* primarily designed to serve as a motivating introduction to the design and its features, or as a tutorial for new users. Rather it presents a careful and fully explicit definition of that design, suitable for guiding implementations. For those in search of a step-by-step introduction to the design, the non-normative [\[XML Schema: Primer\]](#) is a much better starting point than this document.

## 1.1. Purpose

The purpose of *XML Schema: Structures* is to define the nature of XML schemas and their component parts, provide an inventory of XML markup constructs with which to represent schemas, and define the application of schemas to XML documents.

The purpose of an *XML Schema: Structures* schema is to define and describe a class of XML documents by using schema components to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content and attributes and their values. Schemas may also provide for the specification of additional document information, such as normalization and defaulting of attribute and element values. Schemas have facilities for self-documentation. Thus, *XML Schema: Structures* can be used to define, describe and catalogue XML vocabularies for classes of XML documents.

Any application that consumes well-formed XML can use the *XML Schema: Structures* formalism to express syntactic, structural and value constraints applicable to its document instances. The *XML Schema: Structures* formalism allows a useful level of constraint checking to be described and implemented for a wide spectrum of XML applications. However, the language defined by this specification does not attempt to provide *all* the facilities that might be needed by *any* application. Some applications may require constraint capabilities not expressible in this language, and so may need to perform their own additional validations.

## 1.2. Dependencies on Other Specifications

The definition of *XML Schema: Structures* depends on the following specifications: [XML-Infoset], [XML-Namespaces], [XPath], and [XML Schemas: Datatypes].

See [Appendix D – Required Information Set Items and Properties \(normative\)](#) on page 119 for a tabulation of the information items and properties specified in [XML-Infoset] which this specification requires as a precondition to schema-aware processing.

## 1.3. Documentation Conventions and Terminology

The section introduces the highlighting and typography as used in this document to present technical material.

Special terms are defined at their point of introduction in the text. For example a *term* is something used with a special meaning. The definition is labeled as such and the term it defines is displayed in boldface. The end of the definition is not specially marked in the displayed or printed text. Uses of defined terms are links to their definitions, set off with middle dots, for instance [term](#).

Non-normative examples are set off in boxes and accompanied by a brief explanation:

 `<schema targetNamespace="http://www.example.com/XMLSchema/1.0/mySchema">`

And an explanation of the example.

The definition of each kind of schema component consists of a list of its properties and their contents, followed by descriptions of the semantics of the properties:


Definition of the property.

References to properties of schema components are links to the relevant definition as exemplified above, set off with curly braces, for instance .

The correspondence between an element information item which is part of the XML representation of a schema and one or more schema components is presented in a tableau which illustrates the element information item(s) involved. This is followed by a tabulation of the correspondence between properties of the component and properties of the information item. Where context may determine which of several different components may arise, several tabulations, one per context, are given. The property correspondences are normative, as are the illustrations of the XML representation element information items.

In the XML representation, bold-face attribute names (e.g. *count* below) indicate a required attribute information item, and the rest are optional. Where an attribute information item has an enumerated type definition, the values are shown separated by vertical bars, as for *size* below; if there is a default value, it is shown following a colon. Where an attribute information item has a built-in simple type definition defined in [XML Schemas: Datatypes], a hyperlink to its definition therein is given.

The allowed content of the information item is shown as a grammar fragment, using the Kleene operators ?, \* and +. Each element name therein is a hyperlink to its own illustration.

 The illustrations are derived automatically from the [Appendix A – Schema for Schemas \(normative\)](#) on page 118. In the case of apparent conflict, the [Appendix A – Schema for Schemas \(normative\)](#) on page 118 takes precedence, as it, together with the [Schema Representation Constraints](#), provide the normative statement of the form of XML representations.

Description of what the property corresponds to, e.g. the value of the *size* attribute

References to elements in the text are links to the relevant illustration as exemplified above, set off with angle brackets, for instance .

References to properties of information items as defined in [\[XML-Infoset\]](#) are notated as links to the relevant section thereof, set off with square brackets, for example children.

Properties which this specification defines for information items are introduced as follows:

The value the property gets.

References to properties of information items defined in this specification are notated as links to their introduction as exemplified above, set off with square brackets, for example .

The following highlighting is used for non-normative commentary in this document:



General comments directed to all readers.

Following [\[XML 1.0 \(Second Edition\)\]](#), within normative prose in this specification, the words *may* and *must* are defined as follows:

### ***may***

Conforming documents and XML Schema-aware processors are permitted to but need not behave as described.

### ***must***

Conforming documents and XML Schema-aware processors are required to behave as described; otherwise they are in error.

Note however that this specification provides a definition of error and of conformant processors' responsibilities with respect to errors (see [§ 5 – Schemas and Schema-validity Assessment](#) on page 115) which is considerably more complex than that of [\[XML 1.0 \(Second Edition\)\]](#).

## **2. Conceptual Framework**

This chapter gives an overview of *XML Schema: Structures* at the level of its abstract data model. [§ 3 – Schema Component Details](#) on page 13 provides details on this model, including a normative representation in XML for the components of the model. Readers interested primarily in learning to write schema documents may wish to first read [\[XML Schema: Primer\]](#) for a tutorial introduction, and only then consult the sub-sections of [§ 3 – Schema Component Details](#) on page 13 named *XML Representation of ...* for the details.

### **2.1. Overview of XML Schema**

An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items (as defined in [\[XML-Infoset\]](#)), and furthermore may specify augmentations to those items and their descendants. This augmentation makes explicit information which may have been implicit in the original document, such as normalized and/or default values for attributes and elements and the types of element and attribute information items. We refer to the augmented infoset which results from conformant processing as defined in this specification as the *post-schema-validation infoset*, or PSVI.

Schema-validity assessment has two aspects:

1. Determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema;
2. Synthesizing an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any, and adding appropriate augmentations to the infoset to record this outcome.

Throughout this specification, the word *valid* and its derivatives are used to refer to above, the determination of local schema-validity.

Throughout this specification, the word *assessment* is used to refer to the overall process of local validation, schema-validity assessment and infoset augmentation.

## 2.2. XML Schema Abstract Data Model

This specification builds on [XML 1.0 (Second Edition)] and [XML-Namespaces]. The concepts and definitions used herein regarding XML are framed at the abstract level of information items as defined in [XML-Infoset]. By definition, this use of the infoset provides *a priori* guarantees of well-formedness (as defined in [XML 1.0 (Second Edition)]) and namespace conformance (as defined in [XML-Namespaces]) for all candidates for *assessment* and for all *schema documents*.

Just as [XML 1.0 (Second Edition)] and [XML-Namespaces] can be described in terms of information items, XML Schemas can be described in terms of an abstract data model. In defining XML Schemas in terms of an abstract data model, this specification rigorously specifies the information which must be available to a conforming XML Schema processor. The abstract model for schemas is conceptual only, and does not mandate any particular implementation or representation of this information. To facilitate interoperability and sharing of schema information, a normative XML interchange format for schemas is provided.

*Schema component* is the generic term for the building blocks that comprise the abstract data model of the schema. An *XML Schema* is a set of *schema components*. There are 13 kinds of component in all, falling into three groups. The primary components, which may (type definitions) or must (element and attribute declarations) have names are as follows:

- Simple type definitions
- Complex type definitions
- Attribute declarations
- Element declarations

The secondary components, which must have names, are as follows:

- Attribute group definitions
- Identity-constraint definitions
- Model group definitions
- Notation declarations

Finally, the “helper” components provide small parts of other components; they are not independent of their context:

- Annotations


- Model groups
- Particles
- Wildcards
- Attribute Uses

During *validation*, *declaration* components are associated by (qualified) name to information items being *validated*.

On the other hand, *definition* components define internal schema components that can be used in other schema components.

Declarations and definitions may have and be identified by *names*, which are NCNames as defined by [XML-Namespaces].

Several kinds of component have a *target namespace*, which is either *absent* or a namespace name, also as defined by [XML-Namespaces]. The *target namespace* serves to identify the namespace within which the association between the component and its name exists. In the case of declarations, this in turn determines the namespace name of, for example, the element information items it may *validate*.

 At the abstract level, there is no requirement that the components of a schema share a *target namespace*. Any schema for use in *assessment* of documents containing names from more than one namespace will of necessity include components with different *target namespaces*. This contrasts with the situation at the level of the XML representation of components, in which each schema document contributes definitions and declarations to a single target namespace.

*Validation*, defined in detail in § 3 – *Schema Component Details* on page 13, is a relation between information items and schema components. For example, an attribute information item may *validate* with respect to an attribute declaration, a list of element information items may *validate* with respect to a content model, and so on. The following sections briefly introduce the kinds of components in the schema abstract data model, other major features of the abstract model, and how they contribute to *validation*.

### 2.2.1. Type Definition Components

The abstract model provides two kinds of type definition component: simple and complex.

This specification uses the phrase *type definition* in cases where no distinction need be made between simple and complex types.

Type definitions form a hierarchy with a single root. The subsections below first describe characteristics of that hierarchy, then provide an introduction to simple and complex type definitions themselves.

#### 2.2.1.1. Type Definition Hierarchy

Except for a distinguished *ur-type definition*, every *type definition* is, by construction, either a *restriction* or an *extension* of some other type definition. The graph of these relationships forms a tree known as the *Type Definition Hierarchy*.

A type definition whose declarations or facets are in a one-to-one relation with those of another specified type definition, with each in turn restricting the possibilities of the one it corresponds to, is said to be a *restriction*. The specific restrictions might include narrowed ranges or reduced alternatives. Members of a type, A, whose definition is a *restriction* of the definition of another type, B, are always members of type B as well.

A complex type definition which allows element or attribute content in addition to that allowed by another specified type definition is said to be an *extension*.

A distinguished complex type definition, the *ur-type definition*, whose name is `anyType` in the XML Schema namespace, is present in each [XML Schema](#), serving as the root of the type definition hierarchy for that schema.


A type definition used as the basis for an [extension](#) or [restriction](#) is known as the *base type definition* of that definition.

### 2.2.1.2. Simple Type Definition

A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the [normalized value](#) of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements.

Each simple type definition, whether built-in (that is, defined in [\[XML Schemas: Datatypes\]](#)) or user-defined, is a [restriction](#) of some particular simple [base type definition](#). For the built-in primitive type definitions, this is the *simple ur-type definition*, a special restriction of the [ur-type definition](#), whose name is `anySimpleType` in the XML Schema namespace. The [simple ur-type definition](#) is considered to have an unconstrained lexical space, and a value space consisting of the union of the value spaces of all the built-in primitive datatypes and the set of all lists of all members of the value spaces of all the built-in primitive datatypes.

The mapping from lexical space to value space is unspecified for items whose type definition is the [simple ur-type definition](#). Accordingly this specification does not constrain processors' behaviour in areas where this mapping is implicated, for example checking such items against enumerations, constructing default attributes or elements whose declared type definition is the [simple ur-type definition](#), checking identity constraints involving such items.

 The Working Group expects to return to this area in a future version of this specification.

Simple types may also be defined whose members are lists of items themselves constrained by some other simple type definition, or whose membership is the union of the memberships of some other simple type definitions. Such list and union simple type definitions are also restrictions of the [simple ur-type definition](#).

For detailed information on simple type definitions, see § 3.14 – [Simple Type Definitions](#) on page 89 and [\[XML Schemas: Datatypes\]](#). The latter also defines an extensive inventory of pre-defined simple types.

### 2.2.1.3. Complex Type Definition

A complex type definition is a set of attribute declarations and a content type, applicable to the attributes and children of an element information item respectively. The content type may require the children to contain neither element nor character information items (that is, to be empty), to be a string which belongs to a particular simple type or to contain a sequence of element information items which conforms to a particular model group, with or without character information items as well.

Each complex type definition other than the [ur-type definition](#) is either

- a restriction of a complex [base type definition](#)
- or
- an [extension](#) of a simple or complex [base type definition](#).

A complex type which extends another does so by having additional content model particles at the end of the other definition's content model, or by having additional attribute declarations, or both.





This specification allows only appending, and not other kinds of extensions. This decision simplifies application processing required to cast instances from derived to base type. Future versions may allow more kinds of extension, requiring more complex transformations to effect casting.

For detailed information on complex type definitions, see [§ 3.4 – Complex Type Definitions](#) on page 38.

## 2.2.2. Declaration Components

There are three kinds of declaration component: element, attribute, and notation. Each is described in a section below. Also included is a discussion of element substitution groups, which is a feature provided in conjunction with element declarations.

### 2.2.2.1. Element Declaration

An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. The association is either global or scoped to a containing complex type definition. A top-level element declaration with name 'A' is broadly comparable to a pair of DTD declarations as follows, where the associated type definition fills in the ellipses:

```
<!ELEMENT A . . . >  
<!ATTLIST A . . . >
```

Element declarations contribute to [validation](#) as part of model group [validation](#), when their defaults and type components are checked against an element information item with a matching name and namespace, and by triggering identity-constraint definition [validation](#).

For detailed information on element declarations, see [§ 3.3 – Element Declarations](#) on page 22.

### 2.2.2.2. Element Substitution Group

In XML 1.0, the name and content of an element must correspond exactly to the element type referenced in the corresponding content model.

Through the new mechanism of *element substitution groups*, XML Schemas provides a more powerful model supporting substitution of one named element for another. Any top-level element declaration can serve as the defining member, or head, for an element substitution group. Other top-level element declarations, regardless of target namespace, can be designated as members of the substitution group headed by this element. In a suitably enabled content model, a reference to the head [validates](#) not just the head itself, but elements corresponding to any other member of the substitution group as well.

All such members must have type definitions which are either the same as the head's type definition or restrictions or extensions of it. Therefore, although the names of elements can vary widely as new namespaces and members of the substitution group are defined, the content of member elements is strictly limited according to the type definition of the substitution group head.

Note that element substitution groups are not represented as separate components. They are specified in the property values for element declarations (see [§ 3.3 – Element Declarations](#) on page 22).

### 2.2.2.3. Attribute Declaration

An attribute declaration is an association between a name and a simple type definition, together with occurrence information and (optionally) a default value. The association is either global, or local to its containing complex type definition. Attribute declarations contribute to [validation](#) as part of complex type

definition [validation](#), when their occurrence, defaults and type components are checked against an attribute information item with a matching name and namespace.

For detailed information on attribute declarations, see [§ 3.2 – Attribute Declarations](#) on page 16.

#### 2.2.2.4. Notation Declaration

A notation declaration is an association between a name and an identifier for a notation. For an attribute information item to be [valid](#) with respect to a NOTATION simple type definition, its value must have been declared with a notation declaration.

For detailed information on notation declarations, see [§ 3.12 – Notation Declarations](#) on page 86.

### 2.2.3. Model Group Components

The model group, particle, and wildcard components contribute to the portion of a complex type definition that controls an element information item's content.

#### 2.2.3.1. Model Group

A model group is a constraint in the form of a grammar fragment that applies to lists of element information items. It consists of a list of particles, i.e. element declarations, wildcards and model groups. There are three varieties of model group:


- Sequence (the element information items match the particles in sequential order);
- Conjunction (the element information items match the particles, in any order);
- Disjunction (the element information items match one of the particles).

For detailed information on model groups, see [§ 3.8 – Model Groups](#) on page 60.

#### 2.2.3.2. Particle

A particle is a term in the grammar for element content, consisting of either an element declaration, a wildcard or a model group, together with occurrence constraints. Particles contribute to [validation](#) as part of complex type definition [validation](#), when they allow anywhere from zero to many element information items or sequences thereof, depending on their contents and occurrence constraints.

A particle can be used in a complex type definition to constrain the [validation](#) of the children of an element information item; such a particle is called a *content model*.

 *XML Schema: Structures* [content models](#) are similar to but more expressive than [XML 1.0 (Second Edition)] content models; unlike [XML 1.0 (Second Edition)], *XML Schema: Structures* applies [content models](#) to the [validation](#) of both mixed and element-only content.

For detailed information on particles, see [§ 3.9 – Particles](#) on page 65.

#### 2.2.3.3. Attribute Use

An attribute use plays a role similar to that of a particle, but for attribute declarations: an attribute declaration within a complex type definition is embedded within an attribute use, which specifies whether the declaration requires or merely allows its attribute, and whether it has a default or fixed value.

#### 2.2.3.4. Wildcard

A wildcard is a special kind of particle which matches element and attribute information items dependent on their namespace name, independently of their local names.



---

For detailed information on wildcards, see [§ 3.10 – Wildcards](#) on page 74.

### 2.2.4. Identity-constraint Definition Components

An identity-constraint definition is an association between a name and one of several varieties of identity-constraint related to uniqueness and reference. All the varieties use [XPath] expressions to pick out sets of information items relative to particular target element information items which are unique, or a key, or a **valid** reference, within a specified scope. An element information item is only **valid** with respect to an element declaration with identity-constraint definitions if those definitions are all satisfied for all the descendants of that element information item which they pick out.

For detailed information on identity-constraint definitions, see [§ 3.11 – Identity-constraint Definitions](#) on page 79.

### 2.2.5. Group Definition Components

There are two kinds of convenience definitions provided to enable the re-use of pieces of complex type definitions: model group definitions and attribute group definitions.

#### 2.2.5.1. Model Group Definition

A model group definition is an association between a name and a model group, enabling re-use of the same model group in several complex type definitions.

For detailed information on model group definitions, see [§ 3.7 – Model Group Definitions](#) on page 58.

#### 2.2.5.2. Attribute Group Definition

An attribute group definition is an association between a name and a set of attribute declarations, enabling re-use of the same set in several complex type definitions.

For detailed information on attribute group definitions, see [§ 3.6 – Attribute Group Definitions](#) on page 56.

### 2.2.6. Annotation Components

An annotation is information for human and/or mechanical consumers. The interpretation of such information is not defined in this specification.

For detailed information on annotations, see [§ 3.13 – Annotations](#) on page 88.

## 2.3. Constraints and Validation Rules

The [XML 1.0 (Second Edition)] specification describes two kinds of constraints on XML documents: *well-formedness* and *validity* constraints. Informally, the well-formedness constraints are those imposed by the definition of XML itself (such as the rules for the use of the < and > characters and the rules for proper nesting of elements), while validity constraints are the further constraints on document structure provided by a particular DTD.

The preceding section focused on **validation**, that is the constraints on information items which schema components supply. In fact however this specification provides four different kinds of normative statements about schema components, their representations in XML and their contribution to the **validation** of information items:

### ***Schema Component Constraint***

Constraints on the schema components themselves, i.e. conditions components must satisfy to be components at all. Located in the sixth sub-section of the per-component sections of § 3 – [Schema Component Details](#) on page 13 and tabulated in [Appendix C.4 – Schema Component Constraints](#) on page 119.

### ***Schema Representation Constraint***

Constraints on the representation of schema components in XML beyond those which are expressed in [Appendix A – Schema for Schemas \(normative\)](#) on page 118. Located in the third sub-section of the per-component sections of § 3 – [Schema Component Details](#) on page 13 and tabulated in [Appendix C.3 – Schema Representation Constraints](#) on page 119.

### ***Validation Rules***

Contributions to [validation](#) associated with schema components. Located in the fourth sub-section of the per-component sections of § 3 – [Schema Component Details](#) on page 13 and tabulated in [Appendix C.1 – Validation Rules](#) on page 119.

### ***Schema Information Set Contribution***

Augmentations to [post-schema-validation infosets](#) expressed by schema components, which follow as a consequence of [validation](#) and/or [assessment](#). Located in the fifth sub-section of the per-component sections of § 3 – [Schema Component Details](#) on page 13 and tabulated in [Appendix C.2 – Contributions to the post-schema-validation infoset](#) on page 119.


The last of these, schema information set contributions, are not as new as they might at first seem. XML 1.0 validation augments the XML 1.0 information set in similar ways, for example by providing values for attributes not present in instances, and by implicitly exploiting type information for normalization or access. (As an example of the latter case, consider the effect of NMTOKENS on attribute white space, and the semantics of ID and IDREF.) By including schema information set contributions, this specification makes explicit some features that XML 1.0 left implicit.

## **2.4. Conformance**

This specification describes three levels of conformance for schema aware processors. The first is required of all processors. Support for the other two will depend on the application environments for which the processor is intended.

*Minimally conforming* processors must completely and correctly implement the [Schema Component Constraints](#), [Validation Rules](#), and [Schema Information Set Contributions](#) contained in this specification.

*Minimally conforming* processors which accept schemas represented in the form of XML documents as described in § 4.2 – [Layer 2: Schema Documents, Namespaces and Composition](#) on page 105 are additionally said to provide *conformance to the XML Representation of Schemas*. Such processors must, when processing schema documents, completely and correctly implement all [Schema Representation Constraints](#) in this specification, and must adhere exactly to the specifications in § 3 – [Schema Component Details](#) on page 13 for mapping the contents of such documents to [schema components](#) for use in [validation](#) and [assessment](#).

 By separating the conformance requirements relating to the concrete syntax of XML schema documents, this specification admits processors which use schemas stored in optimized binary representations, dynamically created schemas represented as programming language data structures, or implementations in which particular schemas are compiled into executable code such as C or Java. Such processors can be said to be [minimally conforming](#) but not necessarily in [conformance to the XML Representation of Schemas](#).

*Fully conforming* processors are network-enabled processors which are not only both [minimally conforming](#) and [in conformance to the XML Representation of Schemas](#), but which additionally must be capable of accessing schema documents from the World Wide Web according to [§ 2.7 – Representation of Schemas on the World Wide Web](#) on page 12 and [§ 4.3.2 – How schema definitions are located on the Web](#) on page 113. .



Although this specification provides just these three standard levels of conformance, it is anticipated that other conventions can be established in the future. For example, the World Wide Web Consortium is considering conventions for packaging on the Web a variety of resources relating to individual documents and namespaces. Should such developments lead to new conventions for representing schemas, or for accessing them on the Web, new levels of conformance can be established and named at that time. There is no need to modify or republish this specification to define such additional levels of conformance.

See [§ 4 – Schemas and Namespaces: Access and Composition](#) on page 103 for a more detailed explanation of the mechanisms supporting these levels of conformance.

## 2.5. Names and Symbol Spaces

As discussed in [§ 2.2 – XML Schema Abstract Data Model](#) on page 4, most schema components (may) have [names](#). If all such names were assigned from the same “pool”, then it would be impossible to have, for example, a simple type definition and an element declaration both with the name “title” in a given [target namespace](#).

Therefore this specification introduces the term *symbol space* to denote a collection of names, each of which is unique with respect to the others. A symbol space is similar to the non-normative concept of namespace partition introduced in [\[XML-Namespaces\]](#). There is a single distinct symbol space within a given [target namespace](#) for each kind of definition and declaration component identified in [§ 2.2 – XML Schema Abstract Data Model](#) on page 4, except that within a target namespace, simple type definitions and complex type definitions share a symbol space. Within a given symbol space, names are unique, but the same name may appear in more than one symbol space without conflict. For example, the same name can appear in both a type definition and an element declaration, without conflict or necessary relation between the two.

Locally scoped attribute and element declarations are special with regard to symbol spaces. Every complex type definition defines its own local attribute and element declaration symbol spaces, where these symbol spaces are distinct from each other and from any of the other symbol spaces. So, for example, two complex type definitions having the same target namespace can contain a local attribute declaration for the unqualified name “priority”, or contain a local element declaration for the name “address”, without conflict or necessary relation between the two.

## 2.6. Schema-Related Markup in Documents Being Validated

The XML representation of schema components uses a vocabulary identified by the namespace name `http://www.w3.org/2001/XMLSchema`. For brevity, the text and examples in this specification use the prefix `xs:` to stand for this namespace; in practice, any prefix can be used.

*XML Schema: Structures* also defines several attributes for direct use in any XML documents. These attributes are in a different namespace, which has the namespace name `http://www.w3.org/2001/XMLSchema-instance`. For brevity, the text and examples in this specification use the prefix `xsi:` to stand for this latter namespace; in practice, any prefix can be used. All schema processors have appropriate attribute declarations for these attributes built in, see Attribute

Declaration for the 'type' attribute type <http://www.w3.org/2001/XMLSchema-instance> The built-in QName simple type definition global absent absent , Attribute Declaration for the 'nil' attribute nil <http://www.w3.org/2001/XMLSchema-instance> The built-in boolean simple type definition global absent absent , Attribute Declaration for the 'schemaLocation' attribute schemaLocation <http://www.w3.org/2001/XMLSchema-instance> An anonymous simple type definition, as follows: absent <http://www.w3.org/2001/XMLSchema-instance> The built in simple ur-type definition absent list The built-in anyURI simple type definition absent global absent absent and Attribute Declaration for the 'noNamespaceSchemaLocation' attribute noNamespaceSchemaLocation <http://www.w3.org/2001/XMLSchema-instance> The built-in anyURI simple type definition global absent absent .

### 2.6.1. xsi:type

The § 2.2.1.2 – Simple Type Definition on page 6 or § 2.2.1.3 – Complex Type Definition on page 6 used in validation of an element is usually determined by reference to the appropriate schema components. An element information item in an instance may, however, explicitly assert its type using the attribute `xsi:type`. The value of this attribute is a QName; see QName Interpretation Where the type of an attribute information item in a document involved in validation is identified as QName, its actual value is composed of a local name and a namespace name. Its actual value is determined based on its normalized value and the containing element information item's in-scope namespaces following : 1. 2. In the absence of the in-scope namespaces property in the infoset for the schema document in question, processors must reconstruct equivalent information as necessary, using the namespace attributes of the containing element information item and its ancestors. for the means by which the QName is associated with a type definition.

### 2.6.2. xsi:nil

*XML Schema: Structures* introduces a mechanism for signaling that an element should be accepted as **valid** when it has no content despite a content type which does not require or even necessarily allow empty content. An element may be **valid** without content if it has the attribute `xsi:nil` with the value `true`. An element so labeled must be empty, but can carry attributes if permitted by the corresponding complex type.

### 2.6.3. xsi:schemaLocation, xsi:noNamespaceSchemaLocation

The `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes can be used in a document to provide hints as to the physical location of schema documents which may be used for assessment. See § 4.3.2 – How schema definitions are located on the Web on page 113 for details on the use of these attributes.

## 2.7. Representation of Schemas on the World Wide Web

On the World Wide Web, schemas are conventionally represented as XML documents (preferably of MIME type `application/xml` or `text/xml`, but see of Inclusion Constraints and Semantics In addition to the conditions imposed on element information items by the schema for schemas, 1. 2. 3. It is not an error for the actual value of the `schemaLocation` attribute to fail to resolve it all, in which case no corresponding inclusion is performed. It is an error for it to resolve but the rest of clause 1 above to fail to be satisfied. Failure to resolve may well cause less than complete assessment outcomes, of course. As discussed in , QNames in XML representations may fail to resolve, rendering components incomplete and unusable because of missing subcomponents. During schema construction, implementations must retain QName values for such references, in case an appropriately-named component becomes available to discharge the reference by the time it is actually needed. Absent target namespace names of such as-yet

unresolved reference QNames in d components must also be converted if is satisfied. ), conforming to the specifications in § 4.2 – Layer 2: Schema Documents, Namespaces and Composition on page 105. For more information on the representation and use of schema documents on the World Wide Web see § 4.3.1 – Standards for representation of schemas and retrieval of schema documents on the Web on page 113 and § 4.3.2 – How schema definitions are located on the Web on page 113.

## 3. Schema Component Details

### 3.1. Introduction

The following sections provide full details on the composition of all schema components, together with their XML representations and their contributions to [assessment](#). Each section is devoted to a single component, with separate subsections for

1. properties: their values and significance
2. XML representation and the mapping to properties
3. constraints on representation
4. validation rules
5. [post-schema-validation info](#)set contributions
6. constraints on the components themselves

The sub-sections immediately below introduce conventions and terminology used throughout the component sections.

#### 3.1.1. Components and Properties

Components are defined in terms of their properties, and each property in turn is defined by giving its range, that is the values it may have. This can be understood as defining a schema as a labeled directed graph, where the root is a schema, every other vertex is a schema component or a literal (string, boolean, number) and every labeled edge is a property. The graph is *not* acyclic: multiple copies of components with the same name in the same [symbol space](#) may not exist, so in some cases re-entrant chains of properties must exist. Equality of components for the purposes of this specification is always defined as equality of names (including target namespaces) within symbol spaces.



A schema and its components as defined in this chapter are an idealization of the information a schema-aware processor requires: implementations are not constrained in how they provide it. In particular, no implications about literal embedding versus indirection follow from the use below of language such as "properties . . . having . . . components as values".

Throughout this specification, the term *absent* is used as a distinguished property value denoting absence.

Any property not identified as optional is required to be present; optional properties which are not present are taken to have [absent](#) as their value. Any property identified as a having a set, subset or list value may have an empty value unless this is explicitly ruled out: this is *not* the same as [absent](#). Any property value identified as a superset or subset of some set may be equal to that set, unless a proper superset or subset is explicitly called for. By 'string' in Part 1 of this specification is meant a sequence of ISO 10646 characters identified as legal XML characters in [[XML 1.0 \(Second Edition\)](#)].

### 3.1.2. XML Representations of Components

The principal purpose of *XML Schema: Structures* is to define a set of schema components that constrain the contents of instances and augment the information sets thereof. Although no external representation of schemas is required for this purpose, such representations will obviously be widely used. To provide for this in an appropriate and interoperable way, this specification provides a normative XML representation for schemas which makes provision for every kind of schema component. A document in this form (i.e. a element information item) is a *schema document*. For the schema document as a whole, and its constituents, the sections below define correspondences between element information items (with declarations in [Appendix A – Schema for Schemas \(normative\)](#) on page 118 and [Appendix G – DTD for Schemas \(non-normative\)](#) on page 120) and schema components. All the element information items in the XML representation of a schema must be in the XML Schema namespace, that is their namespace name must be `http://www.w3.org/2001/XMLSchema`. Although a common way of creating the XML Infosets which are or contain [schema documents](#) will be using an XML parser, this is not required: any mechanism which constructs conformant infosets as defined in [\[XML-Infoset\]](#) is a possible starting point.

Two aspects of the XML representations of components presented in the following sections are constant across them all:

1. All of them allow attributes qualified with namespace names other than the XML Schema namespace itself: these appear as annotations in the corresponding schema component;
2. All of them allow an as their first child, for human-readable documentation and/or machine-targeted information.

### 3.1.3. The Mapping between XML Representations and Components

For each kind of schema component there is a corresponding normative XML representation. The sections below describe the correspondences between the properties of each kind of schema component on the one hand and the properties of information items in that XML representation on the other, together with constraints on that representation above and beyond those implicit in the [Appendix A – Schema for Schemas \(normative\)](#) on page 118.

The language used is as if the correspondences were mappings from XML representation to schema component, but the mapping in the other direction, and therefore the correspondence in the abstract, can always be constructed therefrom.

In discussing the mapping from XML representations to schema components below, the value of a component property is often determined by the value of an attribute information item, one of the attributes of an element information item. Since schema documents are constrained by the [Appendix A – Schema for Schemas \(normative\)](#) on page 118, there is always a simple type definition associated with any such attribute information item. The phrase *actual value* is used to refer to the member of the value space of the simple type definition associated with an attribute information item which corresponds to its [normalized value](#). This will often be a string, but may also be an integer, a boolean, a URI reference, etc. This term is also occasionally used with respect to element or attribute information items in a document being [validated](#).

Many properties are identified below as having other schema components or sets of components as values. For the purposes of exposition, the definitions in this section assume that (unless the property is explicitly identified as optional) all such values are in fact present. When schema components are constructed from XML representations involving reference by name to other components, this assumption may be violated if one or more references cannot be resolved. This specification addresses the matter of missing components in a uniform manner, described in [§ 5.3 – Missing Sub-components](#) on page 117: no mention of handling missing components will be found in the individual component descriptions below.



Forward reference to named definitions and declarations *is* allowed, both within and between [schema documents](#). By the time the component corresponding to an XML representation which contains a forward reference is actually needed for [validation](#) an appropriately-named component may have become available to discharge the reference: see [§ 4 – Schemas and Namespaces: Access and Composition](#) on page 103 for details.

### 3.1.4. White Space Normalization during Validation

Throughout this specification, the *initial value* of some attribute information item is the value of the normalized value property of that item. Similarly, the *initial value* of an element information item is the string composed of, in order, the character code of each character information item in the children of that element information item.

The above definition means that comments and processing instructions, even in the midst of text, are ignored for all [validation](#) purposes.

The *normalized value* of an element or attribute information item is an [initial value](#) whose white space, if any, has been normalized according to the value of the whiteSpace facet of the simple type definition used in its [validation](#):

#### *preserve*

No normalization is done, the value is the [normalized value](#)

#### *replace*

All occurrences of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space).


#### *collapse*

Subsequent to the replacements specified above under *replace*, contiguous sequences of #x20s are collapsed to a single #x20, and initial and/or final #x20s are deleted.

If the simple type definition used in an item's [validation](#) is the [simple ur-type definition](#), the [normalized value](#) must be determined as in the *preserve* case above.

There are three alternative validation rules which may supply the necessary background for the above: Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration [1. 2. 3. 4. \(\)](#), Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition [1. 2. 3. \(\)](#) or Element Locally Valid (Complex Type) For an element information item to be locally valid with respect to a complex type definition [1. 2. 3. 4. 5.](#) When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . ().


These three levels of normalization correspond to the processing mandated in XML 1.0 for element content, CDATA attribute content and tokenized attributed content, respectively. See [Attribute Value Normalization](#) in [\[XML 1.0 \(Second Edition\)\]](#) for the precedent for *replace* and *collapse* for attributes. Extending this processing to element content is necessary to ensure a consistent [validation](#) semantics for simple types, regardless of whether they are applied to attributes or elements. Performing it twice in the case of attributes whose normalized value has already been subject to replacement or collapse on the basis of information in a DTD is necessary to ensure consistent treatment of attributes regardless of the extent to which DTD-based information has been made use of during infoset construction.

 Even when DTD-based information *has* been appealed to, and [Attribute Value Normalization](#) has taken place, the above definition of [normalized value](#) may mean *further* normalization takes place, as for instance when character entity references in attribute values result in white space characters other than spaces in their [initial values](#).

## 3.2. Attribute Declarations

Attribute declarations provide for:

- Local [validation](#) of attribute information item values using a simple type definition;
- Specifying default or fixed values for attribute information items.

 `<xs:attribute name="age" type="xs:positiveInteger" use="required" />`

The XML representation of an attribute declaration.

### 3.2.1. The Attribute Declaration Schema Component

The attribute declaration schema component has the following properties:

An NCName as defined by [\[XML-Namespaces\]](#). Either [absent](#) or a namespace name, as defined in [\[XML-Namespaces\]](#). A simple type definition. Optional. Either global or a complex type definition. Optional. A pair consisting of a value and one of default, fixed. Optional. An annotation.

The property must match the local part of the names of attributes being [validated](#).


The value of the attribute must conform to the supplied .

A non-[absent](#) value of the property provides for [validation](#) of namespace-qualified attribute information items (which must be explicitly prefixed in the character-level form of XML documents). [Absent](#) values of [validate](#) unqualified (unprefixed) items.

A of global identifies attribute declarations available for use in complex type definitions throughout the schema. Locally scoped declarations are available for use only within the complex type definition identified by the property. This property is [absent](#) in the case of declarations within attribute group definitions: their scope will be determined when they are used in the construction of complex type definitions.

reproduces the functions of XML 1.0 default and #FIXED attribute values. default specifies that the attribute is to appear unconditionally in the [post-schema-validation infoset](#), with the supplied value used whenever the attribute is not actually present; fixed indicates that the attribute value if present must equal the supplied constraint value, and if absent receives the supplied value as for default. Note that it is *values* that are supplied and/or checked, not strings.

See § 3.13 – [Annotations](#) on page 88 for information on the role of the property.

 A more complete and formal presentation of the semantics of , and is provided in conjunction with other aspects of complex type [validation](#) (see Element Locally Valid (Complex Type) For an element information item to be locally valid with respect to a complex type definition 1. 2. 3. 4. 5. When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . .)

[\[XML-Infoset\]](#) distinguishes attributes with names such as `xmlns` or `xmlns:xs1` from ordinary attributes, identifying them as namespace attributes. Accordingly, it is unnecessary and in fact not possible for schemas



to contain attribute declarations corresponding to such namespace declarations, see `xmlns` Not Allowed. The `xmlns` of an attribute declaration must not match `xmlns`. The `xmlns` of an attribute is an NCName, which implicitly prohibits attribute declarations of the form `xmlns:*`. No means is provided in this specification to supply a default value for a namespace declaration.

### 3.2.2. XML Representation of Attribute Declaration Schema Components

The XML representation for an attribute declaration schema component is an element information item. It specifies a simple type definition for an attribute either by reference or explicitly, and may provide default information. The correspondences between the properties of the information item and properties of the component are as follows:

If the element information item has as its parent, the corresponding schema component is as follows:

The **actual value** of the `name` attribute The **actual value** of the `targetNamespace` attribute of the parent element information item, or **absent** if there is none. The simple type definition corresponding to the element information item in the children, if present, otherwise the simple type definition to by the **actual value** of the `type` attribute, if present, otherwise the `.global`. If there is a `default` or a `fixed` attribute, then a pair consisting of the **actual value** (with respect to the `.`) of that attribute and either `default` or `fixed`, as appropriate, otherwise **absent**. The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

otherwise if the element information item has or as an ancestor and the `ref` attribute is absent, it corresponds to an attribute use with properties as follows (unless `use='prohibited'`, in which case the item corresponds to nothing at all):

true if the `use` attribute is present with **actual value** required, otherwise false. See the Attribute Declaration mapping immediately below. If there is a `default` or a `fixed` attribute, then a pair consisting of the **actual value** (with respect to the `.` of the `.`) of that attribute and either `default` or `fixed`, as appropriate, otherwise **absent**. The **actual value** of the `name` attribute If `form` is present and its **actual value** is qualified, or if `form` is absent and the **actual value** of `attributeFormDefault` on the ancestor is qualified, then the **actual value** of the `targetNamespace` attribute of the parent element information item, or **absent** if there is none, otherwise **absent**. The simple type definition corresponding to the element information item in the children, if present, otherwise the simple type definition to by the **actual value** of the `type` attribute, if present, otherwise the `.`. If the element information item has as an ancestor, the complex definition corresponding to that item, otherwise (the element information item is within an definition), **absent**. **absent**. The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

otherwise (the element information item has or as an ancestor and the `ref` attribute is present), it corresponds to an attribute use with properties as follows (unless `use='prohibited'`, in which case the item corresponds to nothing at all):

true if the `use` attribute is present with **actual value** required, otherwise false. The (top-level) attribute declaration to by the **actual value** of the `ref` attribute If there is a `default` or a `fixed` attribute, then a pair consisting of the **actual value** (with respect to the `.` of the `.`) of that attribute and either `default` or `fixed`, as appropriate, otherwise **absent**.

Attribute declarations can appear at the top level of a schema document, or within complex type definitions, either as complete (local) declarations, or by reference to top-level declarations, or within attribute group definitions. For complete declarations, top-level or local, the `type` attribute is used when the declaration can use a built-in or pre-declared simple type definition. Otherwise an anonymous is provided inline.

The default when no simple type definition is referenced or provided is the **simple ur-type definition**, which imposes no constraints at all.

Attribute information items [validated](#) by a top-level declaration must be qualified with the of that declaration (if this is [absent](#), the item must be unqualified). Control over whether attribute information items [validated](#) by a local declaration must be similarly qualified or not is provided by the `form` attribute, whose default is provided by the `attributeFormDefault` attribute on the enclosing `<attribute>`, via its determination of `form`.

The names for top-level attribute declarations are in their own [symbol space](#). The names of locally-scoped attribute declarations reside in symbol spaces local to the type definition which contains them.

### 3.2.3. Constraints on XML Representations of Attribute Declarations

#### **src: Attribute Declaration Representation OK**

In addition to the conditions imposed on element information items by the schema for schemas,

1. `default` and `fixed` must not both be present.
2. If `default` and `use` are both present, `use` must have the [actual value](#) optional.
3. If the item's parent is not `<attribute>`, then
  - A. One of `ref` or `name` must be present, but not both.
  - B. If `ref` is present, then all of `form`, `type` and `use` must be absent.
4. `type` and `base` must not both be present.
5. The corresponding attribute declaration must satisfy the conditions set out in [§ 3.2.6 – Constraints on Attribute Declaration Schema Components](#) on page 21.

### 3.2.4. Attribute Declaration Validation Rules

#### **cvc: Attribute Locally Valid**

For an attribute information item to be locally [valid](#) with respect to an attribute declaration

1. The declaration must not be [absent](#) (see [§ 5.3 – Missing Sub-components](#) on page 117 for how this can fail to be the case).
2. Its `form` must not be `absent`.
3. The item's [normalized value](#) must be locally [valid](#) with respect to that as per [String Valid](#) For a string to be locally valid with respect to a simple type definition [1. 2](#). A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the `unparsedEntities` property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .
4. The item's [actual value](#) must match the value of the `value`, if it is present and fixed.

#### **cvc: Schema-Validity Assessment (Attribute)**

The schema-validity assessment of an attribute information item depends on its [validation](#) alone.

During [validation](#), associations between element and attribute information items among the children and attributes on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the *context-determined declarations*. See (in [Element Locally Valid](#)

(Complex Type) For an element information item to be locally valid with respect to a complex type definition 1. 2. 3. 4. 5. When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . ) for attribute declarations, (in Element Sequence Locally Valid (Particle) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle 1. 2. 3. Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. ) for element declarations.

For an attribute information item's schema-validity to have been assessed

1. A non-[absent](#) attribute declaration must be known for it, namely
  - A. A declaration which has been established as its [context-determined declaration](#);
  - B. A declaration resolved to by its local name and namespace name as defined by QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, 1. 2. 3. 4. 5. 6. The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. , provided its [context-determined declaration](#) is not skip.
2. Its [validity](#) with respect to that declaration must have been evaluated as per Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration 1. 2. 3. 4. .
3. Both and of Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration 1. 2. 3. 4. must be satisfied.

For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been *strictly assessed*.

### 3.2.5. Attribute Declaration Information Set Contributions

#### **sic: Assessment Outcome (Attribute)**

If the schema-validity of an attribute information item has been assessed as per Schema-Validity Assessment (Attribute) The schema-validity assessment of an attribute information item depends on its validation alone. During validation, associations between element and attribute information items among the children and attributes on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the context-determined declarations. See (in ) for attribute declarations, (in ) for element declarations. For an attribute information item's schema-validity to have been assessed 1. 2. 3. For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been strictly assessed. , then in the [post-schema-validation info](#)set it has properties as follows:

The nearest ancestor element information item with a property.

1. it was [strictly assessed](#)
  - A. it was [valid](#) as defined by Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration [1. 2. 3. 4.](#)
    - valid;
  - B. [invalid](#).
2. [notKnown](#).
1. it was [strictly assessed](#)
  - full;
2. [none](#).

infoset. See Attribute Default Value For each attribute use in the whose is false and whose is not absent but whose does not match one of the attribute information items in the element information item's attributes as per of above, the post-schema-validation infoset has an attribute information item whose properties are as below added to the attributes of the element information item. local name The 's'. namespace name The 's'. The canonical lexical representation of the effective value constraint value. The canonical lexical representation of the effective value constraint value. The nearest ancestor element information item with a property. [valid](#). [full](#). [schema](#). The added items should also either have (and if appropriate) properties, or their lighter-weight alternatives, as specified in . for the other possible value.

#### **sic: Validation Failure (Attribute)**

If the local [validity](#), as defined by Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration [1. 2. 3. 4.](#) above, of an attribute information item has been assessed, in the [post-schema-validation infoset](#) the item has a property:

1. the item is not [valid](#)
  - a list. Applications wishing to provide information as to the reason(s) for the [validation](#) failure are encouraged to record one or more error codes (see [Appendix C – Outcome Tabulations \(normative\)](#) on page 119) herein.
2. [absent](#).

#### **sic: Attribute Declaration**

If an attribute information item is [valid](#) with respect to an attribute declaration as per Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration [1. 2. 3. 4.](#) then in the [post-schema-validation infoset](#) the attribute information item may, at processor option, have a property:

An [item isomorphic](#) to the declaration component itself.

#### **sic: Attribute Validated by Type**

If of Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration [1. 2. 3. 4.](#) applies with respect to an attribute information item, in the [post-schema-validation infoset](#) the attribute information item has a property:

The [normalized value](#) of the item as [validated](#).

Furthermore, the item has one of the following alternative sets of properties:

Either

An **item isomorphic** to the relevant attribute declaration's component. If and only if that type definition has union, then an **item isomorphic** to that member of its which actually **validated** the attribute item's normalized value.

or

simple. The of the **type definition**. true if the of the **type definition** is **absent**, otherwise false. The of the **type definition**, if it is not **absent**. If it is **absent**, schema processors may, but need not, provide a value unique to the definition.

If the **type definition** has union, then calling that member of the which actually **validated** the attribute item's **normalized value** the *actual member type definition*, there are three additional properties:

The of the **actual member type definition**. true if the of the **actual member type definition** is **absent**, otherwise false. The of the **actual member type definition**, if it is not **absent**. If it is **absent**, schema processors may, but need not, provide a value unique to the definition.

The first (**item isomorphic**) alternative above is provided for applications such as query processors which need access to the full range of details about an item's **assessment**, for example the type hierarchy; the second, for lighter-weight processors for whom representing the significant parts of the type hierarchy as information items might be a significant burden.

Also, if the declaration has a , the item has a property:

The canonical lexical representation of the declaration's value.

If the attribute information item was not **strictly assessed**, then instead of the values specified above,

1. The item's property has the **initial value** of the item as its value;
2. The and properties, or their alternatives, are based on the .

### 3.2.6. Constraints on Attribute Declaration Schema Components


All attribute declarations (see § 3.2 – Attribute Declarations on page 16) must satisfy the following constraints.

#### cos: Attribute Declaration Properties Correct

1. The values of the properties of an attribute declaration must be as described in the property tableau in § 3.2.1 – The Attribute Declaration Schema Component on page 16, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. if there is a , the canonical lexical representation of its value must be **valid** with respect to the as defined in String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .
3. If the is or is derived from ID then there must not be a .


**cos: xmlns Not Allowed**

The of an attribute declaration must not match xmlns.

 The of an attribute is an [NCName](#), which implicitly prohibits attribute declarations of the form xmlns:.\*

**cos: xsi: Not Allowed**

The of an attribute declaration, whether local or top-level, must not match <http://www.w3.org/2001/XMLSchema-instance> (unless it is one of the four built-in declarations given in the next section).

 This reinforces the special status of these attributes, so that they not only *need* not be declared to be allowed in instances, but *must* not be declared. It also removes any temptation to experiment with supplying global or fixed values for e.g. `xsi:type` or `xsi:nil`, which would be seriously misleading, as they would have no effect.

**3.2.7. Built-in Attribute Declarations**


There are four attribute declarations present in every schema by definition:

Attribute Declaration for the 'type' attribute `type` <http://www.w3.org/2001/XMLSchema-instance> The built-in QName simple type definition global [absent](#) [absent](#) Attribute Declaration for the 'nil' attribute `nil` <http://www.w3.org/2001/XMLSchema-instance> The built-in boolean simple type definition global [absent](#) [absent](#) Attribute Declaration for the 'schemaLocation' attribute `schemaLocation` <http://www.w3.org/2001/XMLSchema-instance> An anonymous simple type definition, as follows: [absent](#) <http://www.w3.org/2001/XMLSchema-instance> The built in [absent](#) list The built-in anyURI simple type definition [absent](#) global [absent](#) [absent](#) Attribute Declaration for the 'noNamespaceSchemaLocation' attribute `noNamespaceSchemaLocation` <http://www.w3.org/2001/XMLSchema-instance> The built-in anyURI simple type definition global [absent](#) [absent](#)

**3.3. Element Declarations**

Element declarations provide for:

- Local [validation](#) of element information item values using a type definition;
- Specifying default or fixed values for an element information items;
- Establishing uniquenesses and reference constraint relationships among the values of related elements and attributes;
- Controlling the substitutability of elements through the mechanism of [element substitution groups](#).



```
<xs:element name="PurchaseOrder" type="PurchaseOrderType"/>

<xs:element name="gift">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="birthday" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    <xs:element ref="PurchaseOrder" />
  </xs:sequence>
</xs:complexType>
</xs:element>

```

XML representations of several different types of element declaration

### 3.3.1. The Element Declaration Schema Component

The element declaration schema component has the following properties:

An NCName as defined by [XML-Namespaces]. Either [absent](#) or a namespace name, as defined in [XML-Namespaces]. Either a simple type definition or a complex type definition. Optional. Either global or a complex type definition. Optional. A pair consisting of a value and one of default, fixed. A boolean. A set of constraint definitions. Optional. A top-level element definition. A subset of {extension, restriction}. A subset of {substitution, extension, restriction}. A boolean. Optional. An annotation.

The property must match the local part of the names of element information items being [validated](#).

A of global identifies element declarations available for use in content models throughout the schema. Locally scoped declarations are available for use only within the complex type identified by the property. This property is [absent](#) in the case of declarations within named model groups: their scope is determined when they are used in the construction of complex type definitions.

A non-[absent](#) value of the property provides for [validation](#) of namespace-qualified element information items. [Absent](#) values of [validate](#) unqualified items.

An element information item is [valid](#) if it satisfies the . For such an item, schema information set contributions appropriate to the are added to the corresponding element information item in the [post-schema-validation infoset](#).

If is true, then an element may also be [valid](#) if it carries the namespace qualified attribute with local name `nil` from namespace `http://www.w3.org/2001/XMLSchema-instance` and value `true` (see § 2.6.2 – [xsi:nil](#) on page 12) even if it has no text or element content despite a which would otherwise require content. Formal details of element [validation](#) are described in Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. .

establishes a default or fixed value for an element. If default is specified, and if the element being [validated](#) is empty, then the canonical form of the supplied constraint value becomes the of the [validated](#) element in the [post-schema-validation infoset](#). If fixed is specified, then the element's content must either be empty, in which case fixed behaves as default, or its value must match the supplied constraint value.



The provision of defaults for elements goes beyond what is possible in XML 1.0 DTDs, and does not exactly correspond to defaults for attributes. In particular, an element with a non-empty whose simple type definition includes the empty string in its lexical space will nonetheless never receive that value, because the will override it.

express constraints establishing uniquenesses and reference relationships among the values of related elements and attributes. See § 3.11 – [Identity-constraint Definitions](#) on page 79.

Element declarations are potential members of the substitution group, if any, identified by . Potential membership is transitive but not symmetric; an element declaration is a potential member of any group of which its is a potential member. Actual membership may be blocked by the effects of or , see below.



An empty allows a declaration to be nominated as the of other element declarations having the same or types derived therefrom. The explicit values of rule out element declarations having types which are extensions or restrictions respectively of . If both values are specified, then the declaration may not be nominated as the of any other declaration.

The supplied values for determine whether an element declaration appearing in a [content model](#) will be prevented from additionally [validating](#) elements (a) with an [§ 2.6.1 – xsi:type](#) on page 12 that identifies an extension or restriction of the type of the declared element, and/or (b) from [validating](#) elements which are in the substitution group headed by the declared element. If is empty, then all derived types and substitution group members are allowed.

Element declarations for which is true can appear in content models only when substitution is allowed; such declarations may not themselves ever be used to [validate](#) element content.

See [§ 3.13 – Annotations](#) on page 88 for information on the role of the property.

### 3.3.2. XML Representation of Element Declaration Schema Components

The XML representation for an element declaration schema component is an element information item. It specifies a type definition for an element either by reference or explicitly, and may provide occurrence and default information. The correspondences between the properties of the information item and properties of the component(s) it corresponds to are as follows:

If the element information item has as its parent, the corresponding schema component is as follows:

The [actual value](#) of the name attribute. The [actual value](#) of the `targetNamespace` attribute of the parent element information item, or [absent](#) if there is none. `global`. The type definition corresponding to the or element information item in the children, if either is present, otherwise the type definition to by the [actual value](#) of the `type` attribute, otherwise the of the element declaration to by the [actual value](#) of the `substitutionGroup` attribute, if present, otherwise the . The [actual value](#) of the `nillable` attribute, if present, otherwise false. If there is a `default` or a `fixed` attribute, then a pair consisting of the [actual value](#) (with respect to the , if it is a simple type definition, or the 's , if that is a simple type definition, or else with respect to the built-in string simple type definition) of that attribute and either default or fixed, as appropriate, otherwise [absent](#). A set consisting of the identity-constraint-definitions corresponding to all the , and element information items in the children, if any, otherwise the empty set. The element declaration to by the [actual value](#) of the `substitutionGroup` attribute, if present, otherwise [absent](#). A set depending on the [actual value](#) of the `block` attribute, if present, otherwise on the [actual value](#) of the `blockDefault` attribute of the ancestor element information item, if present, otherwise on the empty string. Call this the EBV (for effective block value). Then the value of this property is

1. the EBV is the empty string  
the empty set;
2. the EBV is `#all`  
{extension, restriction, substitution};
3. a set with members drawn from the set above, each being present or absent depending on whether the [actual value](#) (which is a list) contains an equivalently named item.



Although the `blockDefault` attribute of may include values other than extension, restriction or substitution, those values are ignored in the determination of for element declarations (they *are* used elsewhere).



As for above, but using the `final` and `finalDefault` attributes in place of the `block` and `blockDefault` attributes and with the relevant set being `{extension, restriction}`. The **actual value** of the `abstract` attribute, if present, otherwise false. The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

otherwise if the element information item has or as an ancestor and the `ref` attribute is absent, the corresponding schema components are as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

The **actual value** of the `minOccurs` attribute, if present, otherwise 1. unbounded, if the `maxOccurs` attribute equals unbounded, otherwise the **actual value** of the `maxOccurs` attribute, if present, otherwise 1. A (local) element declaration as given below.

An element declaration as in the first case above, with the exception of its and properties, which are as below:

If `form` is present and its **actual value** is qualified, or if `form` is absent and the **actual value** of `elementFormDefault` on the ancestor is qualified, then the **actual value** of the `targetNamespace` attribute of the parent element information item, or **absent** if there is none, otherwise **absent**. If the element information item has as an ancestor, the complex definition corresponding to that item, otherwise (the element information item is within a named definition), **absent**.

otherwise (the element information item has or as an ancestor and the `ref` attribute is present), the corresponding schema component is as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

The **actual value** of the `minOccurs` attribute, if present, otherwise 1. unbounded, if the `maxOccurs` attribute equals unbounded, otherwise the **actual value** of the `maxOccurs` attribute, if present, otherwise 1. The (top-level) element declaration to by the **actual value** of the `ref` attribute.

corresponds to an element declaration, and allows the type definition of that declaration to be specified either by reference or by explicit inclusion.

s within produce global element declarations; s within or produce either particles which contain global element declarations (if there's a `ref` attribute) or local declarations (otherwise). For complete declarations, top-level or local, the `type` attribute is used when the declaration can use a built-in or pre-declared type definition. Otherwise an anonymous or is provided inline.

Element information items **validated** by a top-level declaration must be qualified with the of that declaration (if this is **absent**, the item must be unqualified). Control over whether element information items **validated** by a local declaration must be similarly qualified or not is provided by the `form` attribute, whose default is provided by the `elementFormDefault` attribute on the enclosing , via its determination of .

As noted above the names for top-level element declarations are in a separate **symbol space** from the symbol spaces for the names of type definitions, so there can (but need not be) a simple or complex type definition with the same name as a top-level element. As with attribute names, the names of locally-scoped element declarations with no reside in symbol spaces local to the type definition which contains them.

Note that the above allows for two levels of defaulting for unspecified type definitions. An with no referenced or included type definition will correspond to an element declaration which has the same type definition as the head of its substitution group if it identifies one, otherwise the **ur-type definition**. This has the important consequence that the minimum valid element declaration, that is, one with only a name attribute and no contents, is also (nearly) the most general, validating any combination of text and element content and allowing any attributes, and providing for recursive validation where possible.

See below at § 3.11.2 – XML Representation of Identity-constraint Definition Schema Components on page 80 for , and .



```
<xs:element name="unconstrained"/>
```

```
<xs:element name="emptyElt">
  <xs:complexType>
    <xs:attribute ...> . . .</xs:attribute>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="contextOne">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="myLocalElement" type="myFirstType"/>
      <xs:element ref="globalElement"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="contextTwo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="myLocalElement" type="mySecondType"/>
      <xs:element ref="globalElement"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The first example above declares an element whose type, by default, is the [ur-type definition](#). The second uses an embedded anonymous complex type definition.

The last two examples illustrate the use of local element declarations. Instances of `myLocalElement` within `contextOne` will be constrained by `myFirstType`, while those within `contextTwo` will be constrained by `mySecondType`.



The possibility that differing attribute declarations and/or content models would apply to elements with the same name in different contexts is an extension beyond the expressive power of a DTD in XML 1.0.



```
<xs:complexType name="facet">
  <xs:complexContent>
    <xs:extension base="xs:annotated">
      <xs:attribute name="value" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="facet" type="xs:facet" abstract="true"/>

<xs:element name="encoding" substitutionGroup="xs:facet">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:facet">
        <xs:sequence>
          <xs:element ref="annotation" minOccurs="0"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

```

    </xs:sequence>
    <xs:attribute name="value" type="xs:encodings"/>
  </xs:restriction>
</xs:complexContent>
</xs:complexType>
</xs:element>

<xs:element name="period" substitutionGroup="xs:facet">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:facet">
        <xs:sequence>
          <xs:element ref="annotation" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="value" type="xs:duration"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:complexType name="datatype">
  <xs:sequence>
    <xs:element ref="facet" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:NCName" use="optional"/>
  . . .
</xs:complexType>

```

An example from a previous version of the schema for datatypes. The `facet` type is defined and the `facet` element is declared to use it. The `facet` element is abstract -- it's *only* defined to stand as the head for a substitution group. Two further elements are declared, each a member of the `facet` substitution group. Finally a type is defined which refers to `facet`, thereby allowing *either* `period` or `encoding` (or any other member of the group).

### 3.3.3. Constraints on XML Representations of Element Declarations

#### src: Element Declaration Representation OK

In addition to the conditions imposed on element information items by the schema for schemas:

1. `default` and `fixed` must not both be present.
2. If the item's parent is not, then
  - A. One of `ref` or `name` must be present, but not both.
  - B. If `ref` is present, then all of `,, , , nillable, default, fixed, form, block` and `type` must be absent, i.e. only `minOccurs, maxOccurs, id` are allowed in addition to `ref`, along with `.`
3. `type` and `either` or are mutually exclusive.
4. The corresponding particle and/or element declarations must satisfy the conditions set out in § 3.3.6 – [Constraints on Element Declaration Schema Components](#) on page 36 and § 3.9.6 – [Constraints on Particle Schema Components](#) on page 68.

### 3.3.4. Element Declaration Validation Rules

#### **cvc: Element Locally Valid (Element)**

For an element information item to be locally **valid** with respect to an element declaration

1. The declaration must not be **absent**.
2. Its must be false.
3. A. is false
 

there must be no attribute information item among the element information item's attributes whose namespace name is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose local name is `nil`.

B. is true and there is such an attribute information item and its **actual value** is true

  - i. The element information item must have no character or element information item children.
  - ii. There must be no fixed .
4. If there is an attribute information item among the element information item's attributes whose namespace name is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose local name is `type`, then
  - A. The **normalized value** of that attribute information item must be **valid** with respect to the built-in QName simple type, as defined by String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. ;
  - B. The **local name** and **namespace name** (as defined in QName Interpretation Where the type of an attribute information item in a document involved in validation is identified as QName, its actual value is composed of a local name and a namespace name. Its actual value is determined based on its normalized value and the containing element information item's in-scope namespaces following : 1. 2. In the absence of the in-scope namespaces property in the infoset for the schema document in question, processors must reconstruct equivalent information as necessary, using the namespace attributes of the containing element information item and its ancestors. ), of the **actual value** of that attribute information item must resolve to a type definition, as defined in QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, 1. 2. 3. 4. 5. 6. The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. -- call this type definition the *local type definition*;
  - C. The **local type definition** must be validly derived from the given the union of the and the 's , as defined in Type Derivation OK (Complex) For a complex type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction} 1. 2. (if it is a complex type definition), or given as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type

definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. (if it is a simple type definition).

The phrase *actual type definition* occurs below. If the above three clauses are satisfied, this should be understood as referring to the [local type definition](#), otherwise to the .

5. A. the declaration has a , the item has neither element nor character children and has not applied
  - i. If the [actual type definition](#) is a [local type definition](#) then the canonical lexical representation of the value must be a valid default for the [actual type definition](#) as defined in Element Default Valid (Immediate) For a string to be a valid default with respect to a type definition 1. 2. .
  - ii. The element information item with the canonical lexical representation of the value used as its [normalized value](#) must be [valid](#) with respect to the [actual type definition](#) as defined by Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. .
  
- B. the declaration has no or the item has either element or character children or has applied
  - i. The element information item must be [valid](#) with respect to the [actual type definition](#) as defined by Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. .
  - ii. If there is a fixed and has not applied,
    - a) The element information item must have no element information item children.
    - b) 1) the of the [actual type definition](#) is mixed  
the [initial value](#) of the item must match the canonical lexical representation of the value.
    - 2) the of the [actual type definition](#) is a simple type definition  
the [actual value](#) of the item must match the canonical lexical representation of the value.
  
6. The element information item must be [valid](#) with respect to each of the as per Identity-constraint Satisfied For an element information item to be locally valid with respect to an identity-constraint 1. 2. 3. 4. The use of schema normalized value in the definition of key sequence above means that default or fixed value constraints may play a part in key sequences. .
  
7. If the element information item is the [validation root](#), it must be [valid](#) per Validation Root Valid (ID/IDREF) For an element information item which is the validation root to be valid 1. 2. See for the definition of ID/IDREF binding. The first clause above applies when there is a reference to an undefined ID. The second applies when there is a multiply-defined ID. They are separated out to ensure that distinct error codes (see ) are associated with these two cases. Although this rule applies at the validation root, in practice processors, particularly streaming processors, may wish to detect and signal the case as it arises. This reconstruction of 's ID/IDREF functionality is imperfect in that if the validation root is not the document element of an XML document, the results will not necessarily be the same as those a validating parser would give were the document to have a DTD with equivalent declarations. .

**cvc: Element Locally Valid (Type)**

For an element information item to be locally **valid** with respect to a type definition

1. The type definition must not be **absent**;
2. It must not have with value true.
3. A. the type definition is a simple type definition
  - i. The element information item's attributes must be empty, excepting those whose namespace name is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose local name is one of `type`, `nil`, `schemaLocation` or `noNamespaceSchemaLocation`.
  - ii. The element information item must have no element information item children.
  - iii. If of Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. did not apply, then the **normalized value** must be **valid** with respect to the type definition as defined by String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .
- B. the type definition is a complex type definition

the element information item must be **valid** with respect to the type definition as per Element Locally Valid (Complex Type) For an element information item to be locally valid with respect to a complex type definition 1. 2. 3. 4. 5. When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . ;




**cvc: Validation Root Valid (ID/IDREF)**

For an element information item which is the **validation root** to be **valid**

1. There must be no ID/IDREF binding in the item's whose is the empty set.
2. There must be no ID/IDREF binding in the item's whose has more than one member.

See ID/IDREF Table In the post-schema-validation infoset a set of ID/IDREF binding information items is associated with the validation root element information item: A (possibly empty) set of ID/IDREF binding information items, as specified below. Let the eligible item set be the set of consisting of every attribute or element information item for which 1. 2. Then there is one ID/IDREF binding in the for every distinct string which is 1. 2. Each ID/IDREF binding has properties as follows: The string identified above. A set consisting of every element information item for which 1. 2. The net effect of the above is to have one entry for every string used as an id, whether by declaration or by reference, associated with those elements, if any, which actually purport to have that id. See above for the validation rule which actually checks for errors here. The ID/IDREF binding information item, unlike most other aspects of this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of above. Accordingly, conformant processors may, but are not required to, expose it in the post-schema-

validation info set. In other words, the above constraint may be read as saying assessment proceeds as if such an info set item existed. for the definition of ID/IDREF binding.

-  The first clause above applies when there is a reference to an undefined ID. The second applies when there is a multiply-defined ID. They are separated out to ensure that distinct error codes (see [Appendix C – Outcome Tabulations \(normative\)](#) on page 119) are associated with these two cases.
-  Although this rule applies at the [validation root](#), in practice processors, particularly streaming processors, may wish to detect and signal the case as it arises.
-  This reconstruction of [[XML 1.0 \(Second Edition\)](#)]'s ID/IDREF functionality is imperfect in that if the [validation root](#) is not the document element of an XML document, the results will not necessarily be the same as those a validating parser would give were the document to have a DTD with equivalent declarations.

### **cvc: Schema-Validity Assessment (Element)**

The schema-validity assessment of an element information item depends on its [validation](#) and the [assessment](#) of its element information item children and associated attribute information items, if any.

So for an element information item's schema-validity to be assessed

1. A. i. A non-[absent](#) element declaration must be known for it, because
  - a) A declaration was stipulated by the processor (see [§ 5.2 – Assessing Schema-Validity](#) on page 116).
  - b) A declaration has been established as its [context-determined declaration](#).
  - c) 1) Its [context-determined declaration](#) is not skip.
    - 2) Its local name and namespace name resolve to an element declaration as defined by QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, [1. 2. 3. 4. 5. 6.](#) The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. .
  - ii. Its [validity](#) with respect to that declaration must have been evaluated as per Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration [1. 2. 3. 4. 5. 6. 7. .](#)
  - iii. If that evaluation involved the evaluation of Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition [1. 2. 3. .](#), thereof must be satisfied.
- B. i. A non-[absent](#) type definition is known for it because
  - a) A type definition was stipulated by the processor (see [§ 5.2 – Assessing Schema-Validity](#) on page 116).



- b)
    - 1) There is an attribute information item among the element information item's attributes whose namespace name is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose local name is `type`.
    - 2) The **normalized value** of that attribute information item is **valid** with respect to the built-in QName simple type, as defined by String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .
    - 3) The **local name** and **namespace name** (as defined in QName Interpretation Where the type of an attribute information item in a document involved in validation is identified as QName, its actual value is composed of a local name and a namespace name. Its actual value is determined based on its normalized value and the containing element information item's in-scope namespaces following : 1. 2. In the absence of the in-scope namespaces property in the infoset for the schema document in question, processors must reconstruct equivalent information as necessary, using the namespace attributes of the containing element information item and its ancestors. ), of the **actual value** of that attribute information item resolve to a type definition, as defined in QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, 1. 2. 3. 4. 5. 6. The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. -- call this type definition the *local type definition*.
    - 4) If there is also a processor-stipulated type definition, the **local type definition** must be validly derived from that type definition given its , as defined in Type Derivation OK (Complex) For a complex type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction} 1. 2. (if it is a complex type definition), or given the empty set, as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. (if it is a simple type definition).
  - ii. The element information item's **validity** with respect to the **local type definition** (if present and validly derived) or the processor-stipulated type definition (if no **local type definition** is present) has been evaluated as per Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. .
2. The schema-validity of all the element information items among its children has been assessed as per Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item



cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be *laxly assessed* if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an `xsi:type` attribute is involved, however, takes precedence, as is made clear in . , and the schema-validity of all the attribute information items among its attributes has been assessed as per Schema-Validity Assessment (Attribute) The schema-validity assessment of an attribute information item depends on its validation alone. During validation, associations between element and attribute information items among the children and attributes on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the context-determined declarations. See (in ) for attribute declarations, (in ) for element declarations. For an attribute information item's schema-validity to have been assessed 1. 2. 3. For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been strictly assessed. .

If either case of above holds, the element information item has been *strictly assessed*.

If the item cannot be *strictly assessed*, because neither nor above are satisfied, an element information item's schema validity may be *laxly assessed* if its *context-determined declaration* is not skip by *validating* with respect to the as per Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. .



In general if above holds does not, and vice versa. When an `xsi:type` attribute is involved, however, takes precedence, as is made clear in Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. .



The and properties are not mentioned above because they are checked during particle *validation*, as per Element Sequence Locally Valid (Particle) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle 1. 2. 3. Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. .

### 3.3.5. Element Declaration Information Set Contributions

#### **sic: Assessment Outcome (Element)**

If the schema-validity of an element information item has been assessed as per Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be *laxly assessed* if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an `xsi:type` attribute is involved, however, takes precedence, as is made clear in . , then in the *post-schema-validation infoset* it has properties as follows:

The nearest ancestor element information item with a property (or this element item itself if it has such a property).

1. it was *strictly assessed*

A. i. a) of Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . applied and the item was **valid** as defined by Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. ;

b) of Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . applied and the item was **valid** as defined by Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. .

ii. Neither its children nor its attributes contains an information item (element or attribute respectively) whose validity is invalid.

iii. Neither its children nor its attributes contains an information item (element or attribute respectively) with a **context-determined declaration** of mustFind whose validity is notKnown.

valid;

B. invalid..

2. notKnown.

1. it was **strictly assessed** and neither its children nor its attributes contains an information item (element or attribute respectively) whose validation attempted is not full

full;

2. it was not **strictly assessed** and neither its children nor its attributes contains an information item (element or attribute respectively) whose validation attempted is not none

none;

3. partial.

### **sic: Validation Failure (Element)**

If the local **validity**, as defined by Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. above and/or Element Locally

Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. below, of an element information item has been assessed, in the [post-schema-validation infoset](#) the item has a property:

1. the item is not [valid](#)

a list. Applications wishing to provide information as to the reason(s) for the [validation](#) failure are encouraged to record one or more error codes (see [Appendix C – Outcome Tabulations \(normative\)](#) on page 119) herein.

2. [absent](#).

### sic: Element Declaration

If an element information item is [valid](#) with respect to an element declaration as per Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. then in the [post-schema-validation infoset](#) the element information item must, at processor option, have either:

an [item isomorphic](#) to the declaration component itself

or

true if of Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. above is satisfied, otherwise false

### sic: Element Validated by Type

If an element information item is [valid](#) with respect to a [type definition](#) as per Element Locally Valid (Type) For an element information item to be locally valid with respect to a type definition 1. 2. 3. , in the [post-schema-validation infoset](#) the item has a property:

1. of Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. and Element Default Value If the local validity, as defined by above, of an element information item has been assessed, in the [post-schema-validation infoset](#) the item has a property: 1. 2. above have *not* applied and either the [type definition](#) is a simple type definition or its is a simple type definition

the [normalized value](#) of the item as [validated](#).

2. [absent](#).

Furthermore, the item has one of the following alternative sets of properties:

Either

An [item isomorphic](#) to the [type definition](#) component itself. If and only if that type definition is a simple type definition with union, or a complex type definition whose is a simple type definition with union, then an [item isomorphic](#) to that member of the union's which actually [validated](#) the element item's [normalized value](#).

or

simple or complex, depending on the [type definition](#). The target namespace of the [type definition](#). true if the name of the [type definition](#) is [absent](#), otherwise false. The name of the [type definition](#), if it is not [absent](#). If it is [absent](#), schema processors may, but need not, provide a value unique to the definition.

If the [type definition](#) is a simple type definition or its is a simple type definition, and that type definition has union, then calling that member of the which actually [validated](#) the element item's [normalized value](#) the *actual member type definition*, there are three additional properties:

The of the [actual member type definition](#). true if the of the [actual member type definition](#) is [absent](#), otherwise false. The of the [actual member type definition](#), if it is not [absent](#). If it is [absent](#), schema processors may, but need not, provide a value unique to the definition.

The first ([item isomorphic](#)) alternative above is provided for applications such as query processors which need access to the full range of details about an item's [assessment](#), for example the type hierarchy; the second, for lighter-weight processors for whom representing the significant parts of the type hierarchy as information items might be a significant burden.

Also, if the declaration has a , the item has a property:

The canonical lexical representation of the declaration's value.

Note that if an element is [laxly assessed](#), then the and properties, or their alternatives, are based on the .

### sic: Element Default Value

If the local [validity](#), as defined by Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration [1. 2. 3. 4. 5. 6. 7.](#) above, of an element information item has been assessed, in the [post-schema-validation infoset](#) the item has a property:

1. the item is [valid](#) with respect to an element declaration as per Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration [1. 2. 3. 4. 5. 6. 7.](#) and the is present, but of Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration [1. 2. 3. 4. 5. 6. 7.](#) above is not satisfied and the item has no element or character information item children

schema. Furthermore, the [post-schema-validation infoset](#) has the canonical lexical representation of the value as the item's property.

2. [infoset](#).

### 3.3.6. Constraints on Element Declaration Schema Components

All element declarations (see § 3.3 – [Element Declarations](#) on page 22) must satisfy the following constraint.

#### cos: Element Declaration Properties Correct

1. The values of the properties of an element declaration must be as described in the property tableau in § 3.3.1 – [The Element Declaration Schema Component](#) on page 23, modulo the impact of § 5.3 – [Missing Sub-components](#) on page 117.
2. If there is a , the canonical lexical representation of its value must be [valid](#) with respect to the as defined in Element Default Valid (Immediate) For a string to be a valid default with respect to a type definition [1. 2. .](#)
3. If there is a non-[absent](#) , then must be global.
4. If there is a , the of the element declaration must be validly derived from the of the , given the value of the of the , as defined in Type Derivation OK (Complex) For a complex type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of

{extension, restriction} 1. 2. (if the is complex) or as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. (if the is simple).

5. If the or 's is or is derived from ID then there must not be a .



The use of ID as a type definition for elements goes beyond XML 1.0, and should be avoided if backwards compatibility is desired.

6. Circular substitution groups are disallowed. That is, it must not be possible to return to an element declaration by repeatedly following the property.

The following constraints define relations appealed to elsewhere in this specification.

#### cos: Element Default Valid (Immediate)

For a string to be a valid default with respect to a type definition

1. the type definition is a simple type definition

the string must be **valid** with respect to that definition as defined by String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .

2. the type definition is a complex type definition

A. its must be a simple type definition or mixed.

- B. i. the is a simple type definition

the string must be **valid** with respect to that simple type definition as defined by String Valid For a string to be locally valid with respect to a simple type definition 1. 2. A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .

- ii. the is mixed

the 's particle must be **emptiable** as defined by Particle Emptiable For a particle to be emptiable 1. 2. .

#### cos: Substitution Group OK (Transitive)

For an element declaration (call it D) to be validly substitutable for another element declaration (call it C) subject to a blocking constraint (a subset of {substitution, extension, restriction}, the value of a )

1. D and C are the same element declaration.
2. A. The blocking constraint does not contain substitution.
  - B. There is a chain of s from D to C, that is, either D's is C, or D's 's is C, or . . .

- C. The set of all *s* involved in the derivation of *D*'s from *C*'s does not intersect with the union of the blocking constraint, *C*'s (if *C* is complex, otherwise the empty set) and the (respectively the empty set) of any intermediate *s* in the derivation of *D*'s from *C*'s .

### cos: Substitution Group

Every element declaration (call this HEAD) in the of a schema defines a *substitution group*, a subset of those , as follows:

Define *P*, the potential substitution group for HEAD, as follows:

1. The element declaration itself is in *P*;
2. *P* is closed with respect to , that is, if any element declaration in the has a in *P*, then that element is also in *P* itself.

HEAD's actual **substitution group** is then the set consisting of each member of *P* such that

1. Its is false.
2. It is validly substitutable for HEAD subject to HEAD's as the blocking constraint, as defined in Substitution Group OK (Transitive) For an element declaration (call it *D*) to be validly substitutable for another element declaration (call it *C*) subject to a blocking constraint (a subset of {substitution, extension, restriction}, the value of a ) 1. 2. .

## 3.4. Complex Type Definitions

Complex Type Definitions provide for:

- Constraining element information items by providing § 2.2.2.3 – [Attribute Declaration](#) on page 7s governing the appearance and content of attributes
- Constraining element information item children to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character information item children to conform to a specified simple type definition.
- Using the mechanisms of § 2.2.1.1 – [Type Definition Hierarchy](#) on page 5 to derive a complex type from another simple or complex type.
- Specifying [post-schema-validation info set contributions](#) for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.



```
<xs:complexType name="PurchaseOrderType">
  <xs:sequence>
    <xs:element name="shipTo" type="USAddress"/>
    <xs:element name="billTo" type="USAddress"/>
    <xs:element ref="comment" minOccurs="0"/>
    <xs:element name="items" type="Items"/>
  </xs:sequence>
```

```
<xs:attribute name="orderDate" type="xs:date"/>
</xs:complexType>
```

The XML representation of a complex type definition.

### 3.4.1. The Complex Type Definition Schema Component

A complex type definition schema component has the following properties:

Optional. An NCName as defined by [XML-Namespaces]. Either **absent** or a namespace name, as defined in [XML-Namespaces]. Either a simple type definition or a complex type definition. Either extension or restriction. A subset of {extension, restriction}. A boolean A set of attribute uses. Optional. A wildcard. One of empty, a simple type definition or a pair consisting of a **content model** (I.e. a § 2.2.3.2 – Particle on page 8) and one of mixed, element-only. A subset of {extension, restriction}. A set of annotations. Complex types definitions are identified by their and . Except for anonymous complex type definitions (those with no ), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an **XML Schema**, no complex type definition can have the same name as another simple or complex type definition. Complex type **s** and **s** are provided for reference from instances (see § 2.6.1 –  **xsi:type** on page 12), and for use in the XML representation of schema components (specifically in ). See § 4.2.3 – **References to schema components across namespaces** on page 110 for the use of component identifiers when importing one schema into another.



The of a complex type is not *ipso facto* the (local) name of the element information items **validated** by that definition. The connection between a name and a type definition is described in § 3.3 – **Element Declarations** on page 22.

As described in § 2.2.1.1 – **Type Definition Hierarchy** on page 5, each complex type is derived from a which is itself either a § 2.2.1.2 – **Simple Type Definition** on page 6 or a § 2.2.1.3 – **Complex Type Definition** on page 6. specifies the means of derivation as either extension or restriction (see § 2.2.1.1 – **Type Definition Hierarchy** on page 5).

A complex type with an empty specification for can be used as a for other types derived by either of extension or restriction; the explicit values extension, and restriction prevent further derivations by extension and restriction respectively. If all values are specified, then the complex type is said to be *final*, because no further derivations are possible. Finality is *not* inherited, that is, a type definition derived by restriction from a type definition which is final for extension is not itself, in the absence of any explicit final attribute of its own, final for anything.

Complex types for which is true must not be used as the for the **validation** of element information items. It follows that they must not be referenced from an § 2.6.1 –  **xsi:type** on page 12 attribute in an instance document. Abstract complex types can be used as **s**, or even as the **s** of element declarations, provided in every case a concrete derived type definition is used for **validation**, either via § 2.6.1 –  **xsi:type** on page 12 or the operation of a substitution group.

are a set of attribute uses. See **Element Locally Valid (Complex Type)** For an element information item to be locally valid with respect to a complex type definition 1. 2. 3. 4. 5. When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . and **Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration** 1. 2. 3. 4. for details of attribute **validation**.



s provide a more flexible specification for [validation](#) of attributes not explicitly included in . Informally, the specific values of are interpreted as follows:

- any: attributes can include attributes with any qualified or unqualified name.
- a set whose members are either namespace names or [absent](#): attributes can include any attribute(s) from the specified namespace(s). If [absent](#) is included in the set, then any unqualified attributes are (also) allowed.
- 'not' and a namespace name: attributes cannot include attributes from the specified namespace.
- 'not' and [absent](#): attributes cannot include unqualified attributes.

See [Element Locally Valid \(Complex Type\)](#) For an element information item to be locally valid with respect to a complex type definition [1](#). [2](#). [3](#). [4](#). [5](#). When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . and Wildcard allows Namespace Name For a value which is either a namespace name or absent to be valid with respect to a wildcard constraint (the value of a ) [1](#). [2](#). [3](#). for formal details of attribute wildcard [validation](#).

determines the [validation](#) of children of element information items. Informally:

- A with the distinguished value empty [validates](#) elements with no character or element information item children.
- A which is a [§ 2.2.1.2 – Simple Type Definition](#) on page 6 [validates](#) elements with character-only children.
- An element-only [validates](#) elements with children that conform to the supplied [content model](#).
- A mixed [validates](#) elements whose element children (i.e. specifically ignoring other children such as character information items) conform to the supplied [content model](#).

determine whether an element declaration appearing in a [content model](#) is prevented from additionally [validating](#) element items with an [§ 2.6.1 – xsi:type](#) on page 12 attribute that identifies a complex type definition derived by extension or restriction from this definition, or element items in a substitution group whose type definition is similarly derived: If is empty, then all such substitutions are allowed, otherwise, the derivation method(s) it names are disallowed.

See [§ 3.13 – Annotations](#) on page 88 for information on the role of the property.

### 3.4.2. XML Representation of Complex Type Definitions

The XML representation for a complex type definition schema component is a element information item.

The XML representation for complex type definitions with a simple type definition is significantly different from that of those with other s, and this is reflected in the presentation below, which displays first the elements involved in the first case, then those for the second. The property mapping is shown once for each case.

Whichever alternative for the content of is chosen, the following property mappings apply:

The [actual value](#) of the name attribute if present, otherwise [absent](#). The [actual value](#) of the targetNamespace attribute of the ancestor element information item if present, otherwise [absent](#). The [actual value](#) of the abstract attribute, if present, otherwise false. A set corresponding to the [actual value](#) of the block

attribute, if present, otherwise on the [actual value](#) of the `blockDefault` attribute of the ancestor element information item, if present, otherwise on the empty string. Call this the EBV (for effective block value). Then the value of this property is

1. the EBV is the empty string  
the empty set;
2. the EBV is `#all`  
{extension, restriction};
3. a set with members drawn from the set above, each being present or absent depending on whether the [actual value](#) (which is a list) contains an equivalently named item.



Although the `blockDefault` attribute of may include values other than restriction or extension, those values are ignored in the determination of for complex type definitions (they *are* used elsewhere).

As for above, but using the `final` and `finalDefault` attributes in place of the `block` and `blockDefault` attributes. The annotations corresponding to the element information item in the children, if present, in the and children, if present, and in their and children, if present, otherwise [absent](#).

When the alternative is chosen, the following elements are relevant, and the remaining property mappings are as below. Note that either or must be chosen as the content of .

The type definition to by the [actual value](#) of the `base` attribute If the alternative is chosen, then restriction, otherwise (the alternative is chosen) extension. A union of sets of attribute uses as follows

1. The set of attribute uses corresponding to the children, if any.
2. The of the attribute groups to by the [actual values](#) of the `ref` attribute of the children, if any.
3. if the type definition to by the [actual value](#) of the `base` attribute is a complex type definition, the of that type definition, unless the alternative is chosen, in which case some members of that type definition's may not be included, namely those whose 's and are the same as
  - A. the and of the of an attribute use in the set per or above;
  - B. what would have been the and of the of an attribute use in the set per above but for the [actual value](#) of the use attribute of the relevant among the children of being prohibited.

1. Let the *local wildcard* be defined as

- A. there is an present

a wildcard based on the [actual values](#) of the `namespace` and `processContents` attributes and the children, exactly as for the wildcard corresponding to an element as set out in [§ 3.10.2 – XML Representation of Wildcard Schema Components](#) on page 75;

- B. [absent](#).

2. Let the *complete wildcard* be defined as

- A. there are no children corresponding to attribute groups with non-[absent](#) s  
the [local wildcard](#).

- B. there are one or more children corresponding to attribute groups with non-[absent](#) s

i. there is an present

a wildcard whose and are those of the [local wildcard](#), and whose is the intensional intersection of the of the [local wildcard](#) and of the s of all the non-[absent](#) s of the attribute groups corresponding to the children, as defined in Attribute Wildcard Intersection For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2): [1](#). [2](#). [3](#). [4](#). [5](#). [6](#). In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required. .

ii. there is no present

a wildcard whose properties are as follows:

The of the first non-[absent](#) of an attribute group among the attribute groups corresponding to the children.

The intensional intersection of the s of all the non-[absent](#) s of the attribute groups corresponding to the children, as defined in Attribute Wildcard Intersection For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2): [1](#). [2](#). [3](#). [4](#). [5](#). [6](#). In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required. .

[absent](#).

3. The value is then determined by

A. the alternative is chosen

the [complete wildcard](#);

B. the alternative is chosen

i. let the *base wildcard* be defined as

a) the type definition to by the [actual value](#) of the *base* attribute is a complex type definition with an

that .

b) [absent](#).

ii. The value is then determined by

a) the [base wildcard](#) is non-[absent](#)

1) the [complete wildcard](#) is [absent](#)

the [base wildcard](#).

2) a wildcard whose and are those of the [complete wildcard](#), and whose is the intensional union of the of the [complete wildcard](#) and of the [base wildcard](#), as defined in Attribute Wildcard Union For a wildcard's value to be the intensional union of two other such

values (call them O1 and O2): 1. 2. 3. 4. 5. 6. In the case where there are more than two values, the intensional union is determined by identifying the intensional union of two of the values as above, then the intensional union of that value with the third (providing the first union was expressible), and so on as required. .

b) (the [base wildcard is absent](#)) the [complete wildcard](#)

1. the type definition to by the [actual value](#) of the `base` attribute is a complex type definition whose own is a simple type definition and the alternative is chosen

starting from either

- A. the simple type definition corresponding to the among the children of if there is one;
- B. otherwise ( has no among its children), the simple type definition which is the of the type definition to by the [actual value](#) of the `base` attribute

a simple type definition which restricts the simple type definition identified in or with a set of facet components corresponding to the appropriate element information items among the 's children (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S) 1. 2. 3. If above holds, the of R constitute a restriction of the of B with respect to S. ;

2. the type definition to by the [actual value](#) of the `base` attribute is a complex type definition whose own is mixed and a particle which is [emptiable](#), as defined in Particle Emptiable For a particle to be emptiable 1. 2. and the alternative is chosen

starting from the simple type definition corresponding to the among the children of (which must be present) a simple type definition which restricts that simple type definition with a set of facet components corresponding to the appropriate element information items among the 's children (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S) 1. 2. 3. If above holds, the of R constitute a restriction of the of B with respect to S. ;

3. the type definition to by the [actual value](#) of the `base` attribute is a complex type definition (whose own must be a simple type definition, see below) and the alternative is chosen

the of that complex type definition;

4. (the type definition to by the [actual value](#) of the `base` attribute is a simple type definition and the alternative is chosen), then that simple type definition.

When the alternative is chosen, the following elements are relevant (as are the and elements, not repeated here), and the additional property mappings are as below. Note that either or must be chosen as the content of , but their content models are different in this case from the case above when they occur as children of .

The property mappings below are *also* used in the case where the third alternative (neither nor ) is chosen. This case is understood as shorthand for complex content restricting the [ur-type definition](#), and the details of the mappings should be modified as necessary.

The type definition to by the [actual value](#) of the `base` attribute If the alternative is chosen, then restriction, otherwise (the alternative is chosen) extension. A union of sets of attribute uses as follows:

1. The set of attribute uses corresponding to the children, if any.

2. The of the attribute groups to by the **actual values** of the `ref` attribute of the children, if any.
3. The of the type definition to by the **actual value** of the `base` attribute, unless the alternative is chosen, in which case some members of that type definition's may not be included, namely those whose 's and are the same as
  - A. The and of the of an attribute use in the set per or above;
  - B. what would have been the and of the of an attribute use in the set per above but for the **actual value** of the use attribute of the relevant among the children of being prohibited.

As above for the alternative.

1. Let the *effective mixed* be
  - A. the `mixed` attribute is present on its **actual value**;
  - B. the `mixed` attribute is present on its **actual value**;
  - C. `false`.
2. Let the *effective content* be
  - A.
    - i. There is no `,` `,` or among the children;
    - ii. There is an or among the children with no children of its own excluding ;
    - iii. There is a among the children with no children of its own excluding whose `minOccurs` attribute has the **actual value** 0;
    - i. the **effective mixed** is `true`

A particle whose properties are as follows:


      - 1
      - 1


A model group whose is sequence and whose is empty.

.
    - ii. empty
  - B. the particle corresponding to the `,` `,` or among the children.
3. Then the value of the property is
  - A. the alternative is chosen
    - i. the **effective content** is empty
      - empty;
      - ii. a pair consisting of


- a) mixed if the [effective mixed](#) is true, otherwise elementOnly
  - b) The [effective content](#).
- B. the alternative is chosen
- i. the [effective content](#) is empty  
the of the type definition to by the [actual value](#) of the base attribute
  - ii. the type definition to by the [actual value](#) of the base attribute has a of empty  
a pair as per above;
  - iii. a pair of mixed or elementOnly (determined as per above) and a particle whose properties are as follows:
    - 1
    - 1

A model group whose is sequence and whose are the particle of the of the type definition to by the [actual value](#) of the base attribute followed by the [effective content](#).

 Aside from the simple coherence requirements enforced above, constraining type definitions identified as restrictions to actually *be* restrictions, that is, to [validate](#) a subset of the items which are [validated](#) by their base type definition, is enforced in [§ 3.4.6 – Constraints on Complex Type Definition Schema Components](#) on page 51.

 The *only* substantive function of the value prohibited for the use attribute of an is in establishing the correspondence between a complex type defined by restriction and its XML representation. It serves to prevent inheritance of an identically named attribute use from the . Such an does not correspond to any component, and hence there is no interaction with either explicit or inherited wildcards in the operation of [§ 3.4.4 – Complex Type Definition Validation Rules](#) on page 48 or [§ 3.4.6 – Constraints on Complex Type Definition Schema Components](#) on page 51.

Careful consideration of the above concrete syntax reveals that a type definition need consist of no more than a name, i.e. that `<complexType name="anything" />` is allowed.



```
<xs:complexType name="length1">
  <xs:simpleContent>
    <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="unit" type="xs:NMTOKEN"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:element name="width" type="length1"/>
```

```
<width unit="cm">25</width>
```

```
<xs:complexType name="length2">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
```

```

    <xs:sequence>
      <xs:element name="size" type="xs:nonNegativeInteger"/>
      <xs:element name="unit" type="xs:NMTOKEN"/>
    </xs:sequence>
  </xs:restriction>
</xs:complexContent>
</xs:complexType>

```

```
<xs:element name="depth" type="length2"/>
```

```

<depth>
  <size>25</size><unit>cm</unit>
</depth>

```

```

<xs:complexType name="length3">
  <xs:sequence>
    <xs:element name="size" type="xs:nonNegativeInteger"/>
    <xs:element name="unit" type="xs:NMTOKEN"/>
  </xs:sequence>
</xs:complexType>

```

Three approaches to defining a type for length: one with character data content constrained by reference to a built-in datatype, and one attribute, the other two using two elements. `length3` is the abbreviated alternative to `length2`: they correspond to identical type definition components.



```

<xs:complexType name="personName">
  <xs:sequence>
    <xs:element name="title" minOccurs="0"/>
    <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="surname"/>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="extendedName">
  <xs:complexContent>
    <xs:extension base="personName">
      <xs:sequence>
        <xs:element name="generation" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

<xs:element name="addressee" type="extendedName"/>

<addressee>
  <forename>Albert</forename>
  <forename>Arnold</forename>
  <surname>Gore</surname>
  <generation>Jr</generation>
</addressee>

```



A type definition for personal names, and a definition derived by extension which adds a single element; an element declaration referencing the derived definition, and a [valid](#) instance thereof.

```

<xs:complexType name="simpleName">
  <xs:complexContent>
    <xs:restriction base="personName">
      <xs:sequence>
        <xs:element name="forename" minOccurs="1" maxOccurs="1"/>
        <xs:element name="surname"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="who" type="simpleName"/>

<who>
  <forename>Bill</forename>
  <surname>Clinton</surname>
</who>

```

A simplified type definition derived from the base type from the previous example by restriction, eliminating one optional daughter and fixing another to occur exactly once; an element declared by reference to it, and a [valid](#) instance thereof.

```

<xs:complexType name="paraType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="emph"/>
    <xs:element ref="strong"/>
  </xs:choice>
  <xs:attribute name="version" type="xs:number"/>
</xs:complexType>

```

A further illustration of the abbreviated form, with the `mixed` attribute appearing on `complexType` itself.

### 3.4.3. Constraints on XML Representations of Complex Type Definitions

#### src: Complex Type Definition Representation OK

In addition to the conditions imposed on element information items by the schema for schemas,

1. If the alternative is chosen, the type definition to by the [actual value](#) of the `base` attribute must be a complex type definition;
2. If the alternative is chosen,
  - A. The type definition to by the [actual value](#) of the `base` attribute must be
    - i. a complex type definition whose is a simple type definition;
    - ii. only if the alternative is also chosen, a complex type definition whose is mixed and a particle which is [emptiable](#), as defined in Particle Emptiable For a particle to be emptiable 1. 2. ;
    - iii. only if the alternative is also chosen, a simple type definition.

B. If above is satisfied, then there must be a among the children of .



Although not explicitly ruled out either here or in [Appendix A – Schema for Schemas \(normative\)](#) on page 118, specifying `<xs:complexType . . .mixed='true'` when the alternative is chosen has no effect on the corresponding component, and should be avoided. This may be ruled out in a subsequent version of this specification.

3. The corresponding complex type definition component must satisfy the conditions set out in [§ 3.4.6 – Constraints on Complex Type Definition Schema Components](#) on page 51;
4. If or in the correspondence specification above for is satisfied, the intensional intersection must be expressible, as defined in [Attribute Wildcard Intersection](#) For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2): [1.](#) [2.](#) [3.](#) [4.](#) [5.](#) [6.](#) In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required. .

### 3.4.4. Complex Type Definition Validation Rules

#### **cvc: Element Locally Valid (Complex Type)**

For an element information item to be locally **valid** with respect to a complex type definition

1. is false.
2. If of [Element Locally Valid \(Element\)](#) For an element information item to be locally valid with respect to an element declaration [1.](#) [2.](#) [3.](#) [4.](#) [5.](#) [6.](#) [7.](#) did not apply, then
  - A. the is empty
 

the element information item has no character or element information item children.
  - B. the is a simple type definition
 

the element information item has no element information item children, and the **normalized value** of the element information item is **valid** with respect to that simple type definition as defined by [String Valid](#) For a string to be locally valid with respect to a simple type definition [1.](#) [2.](#) A string is a declared entity name if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose normalized value the string is. .
  - C. the is element-only
 

the element information item has no character information item children other than those whose character code is defined as a white space in [\[XML 1.0 \(Second Edition\)\]](#)).
  - D. the is element-only or mixed
 

the sequence of the element information item's element information item children, if any, taken in order, is **valid** with respect to the 's particle, as defined in [Element Sequence Locally Valid \(Particle\)](#) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle [1.](#) [2.](#) [3.](#) Clauses and do not interact: an element information item validatable

by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. .

3. For each attribute information item in the element information item's attributes excepting those whose namespace name is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose local name is one of `type`, `nil`, `schemaLocation` or `noNamespaceSchemaLocation`,

- A. there is among the an attribute use with an whose matches the attribute information item's local name and whose is identical to the attribute information item's namespace name (where an `absent` is taken to be identical to a namespace name with no value)

the attribute information must be **valid** with respect to that attribute use as per Attribute Locally Valid (Use) For an attribute information item to be valid with respect to an attribute use its normalized value must match the canonical lexical representation of the attribute use's value, if it is present and fixed. . In this case the of that attribute use is the **context-determined declaration** for the attribute information item with respect to Schema-Validity Assessment (Attribute) The schema-validity assessment of an attribute information item depends on its validation alone. During validation, associations between element and attribute information items among the children and attributes on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the context-determined declarations. See (in ) for attribute declarations, (in ) for element declarations. For an attribute information item's schema-validity to have been assessed 1. 2. 3. For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been strictly assessed. and Assessment Outcome (Attribute) If the schema-validity of an attribute information item has been assessed as per , then in the post-schema-validation info set it has properties as follows: The nearest ancestor element information item with a property. 1. 2. 1. 2. info set. See for the other possible value. .

- B. i. There must be an .
- ii. The attribute information item must be **valid** with respect to it as defined in Item Valid (Wildcard) For an element or attribute information item to be locally valid with respect to a wildcard constraint its namespace name must be valid with respect to the wildcard constraint, as defined in . When this constraint applies 1. 2. 3. .

4. The of each attribute use in the whose is true matches one of the attribute information items in the element information item's attributes as per above.

5. Let the *wild IDs* be the set of all attribute information item to which applied and whose **validation** resulted in a **context-determined declaration** of `mustFind` or no **context-determined declaration** at all, and whose local name and namespace name resolve (as defined by QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, 1. 2. 3. 4. 5. 6. The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. ) to an attribute declaration whose is or is derived from ID. Then

- A. There must be no more than one item in **wild IDs**.

- B. If **wild IDs** is non-empty, there must not be any attribute uses among the whose 's is or is derived from ID.



This clause serves to ensure that even via attribute wildcards no element has more than one attribute of type ID, and that even when an element legitimately lacks a declared attribute of type ID, a wildcard-validated attribute must not supply it. That is, if an element has a type whose attribute declarations include one of type ID, it either has that attribute or no attribute of type ID.



When an is present, this does *not* introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are **assessed**. In such cases the attribute use *always* takes precedence, and the **assessment** of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of .

### 3.4.5. Complex Type Definition Information Set Contributions

#### **sic: Attribute Default Value**

For each attribute use in the whose is false and whose is not **absent** but whose does not match one of the attribute information items in the element information item's attributes as per of Element Locally Valid (Complex Type) For an element information item to be locally valid with respect to a complex type definition 1. 2. 3. 4. 5. When an is present, this does not introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the whose name and target namespace match are assessed. In such cases the attribute use always takes precedence, and the assessment of such items stands or falls entirely on the basis of the attribute use and its . This follows from the details of . above, the **post-schema-validation info**set has an attribute information item whose properties are as below added to the attributes of the element information item.

#### **local name**

The 's .

#### **namespace name**

The 's .

The canonical lexical representation of the **effective value constraint** value.

The canonical lexical representation of the **effective value constraint** value.

The nearest ancestor element information item with a property.

valid.

full.

schema.

The added items should also either have (and if appropriate) properties, or their lighter-weight alternatives, as specified in Attribute Validated by Type If of applies with respect to an attribute information item, in the post-schema-validation info set the attribute information item has a property: The normalized value of the item as validated. Furthermore, the item has one of the following alternative sets of properties: Either An item isomorphic to the relevant attribute declaration's component. If and only if that type definition has union, then an item isomorphic to that member of its which actually validated the attribute item's

normalized value. or simple. The of the type definition. true if the of the type definition is absent, otherwise false. The of the type definition, if it is not absent. If it is absent, schema processors may, but need not, provide a value unique to the definition. If the type definition has union, then calling that member of the which actually validated the attribute item's normalized value the actual member type definition, there are three additional properties: The of the actual member type definition. true if the of the actual member type definition is absent, otherwise false. The of the actual member type definition, if it is not absent. If it is absent, schema processors may, but need not, provide a value unique to the definition. The first (item isomorphic) alternative above is provided for applications such as query processors which need access to the full range of details about an item's assessment, for example the type hierarchy; the second, for lighter-weight processors for whom representing the significant parts of the type hierarchy as information items might be a significant burden. Also, if the declaration has a , the item has a property: The canonical lexical representation of the declaration's value. If the attribute information item was not strictly assessed, then instead of the values specified above, 1. 2. .

### 3.4.6. Constraints on Complex Type Definition Schema Components

All complex type definitions (see § 3.4 – Complex Type Definitions on page 38) must satisfy the following constraints.

#### cos: Complex Type Definition Properties Correct

1. The values of the properties of a complex type definition must be as described in the property tableau in § 3.4.1 – The Complex Type Definition Schema Component on page 39, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. If the is a simple type definition, the must be extension.
3. Circular definitions are disallowed, except for the . That is, it must be possible to reach the by repeatedly following the .
4. Two distinct attribute declarations in the must not have identical s and s.
5. Two distinct attribute declarations in the must not have s which are or are derived from ID.

#### cos: Derivation Valid (Extension)

If the is extension,

1. the is a complex type definition
  - A. The of the must not contain extension.
  - B. Its must be a subset of the of the complex type definition itself, that is, for every attribute use in the of the , there must be an attribute use in the of the complex type definition itself whose has the same , and as its attribute declaration.
  - C. If it has an , the complex type definition must also have one, and the base type definition's 's must be a subset of the complex type definition's 's , as defined by Wildcard Subset For a namespace constraint (call it sub) to be an intensional subset of another namespace constraint (call it super) 1. 2. 3. .
  - D.
    - i. The of the and the of the complex type definition itself must be the same simple type definition.
    - ii. The of both the and the complex type definition itself must be empty.

- iii. a) The of the complex type definition itself must specify a particle.
- b) 1) The of the must be empty.
- 2) 1) Both s must be mixed or both must be element-only.
- 2) The particle of the complex type definition must be a **valid extension** of the 's particle, as defined in Particle Valid (Extension) For a particle (call it E, for extension) to be a valid extension of another particle (call it B, for base) 1. 2. .

E. It must in principle be possible to derive the complex type definition in two steps, the first an extension and the second a restriction (possibly vacuous), from that type definition among its ancestors whose is the .



This requirement ensures that nothing removed by a restriction is subsequently added back by an extension. It is trivial to check if the extension in question is the only extension in its derivation, or if there are no restrictions bar the first from the .

Constructing the intermediate type definition to check this constraint is straightforward: simply re-order the derivation to put all the extension steps first, then collapse them into a single extension. If the resulting definition can be the basis for a valid restriction to the desired definition, the constraint is satisfied.

2. the is a simple type definition

- A. The must be the same simple type definition.
- B. The of the must not contain extension.

If this constraint Derivation Valid (Extension) If the is extension, 1. 2. If this constraint holds of a complex type definition, it is a valid extension of its . holds of a complex type definition, it is a *valid extension* of its .

### cos: Derivation Valid (Restriction, Complex)

If the is restriction

1. The must be a complex type definition whose does not contain restriction.
2. For each attribute use (call this R) in the
  - A. there is an attribute use in the of the (call this B) whose has the same and
    - i. a) B's is false.
    - b) R's is true.
  - ii. R's 's must be validly derived from B's given the empty set as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. .
  - iii. Let the *effective value constraint* of an attribute use be its , if present, otherwise its 's . Then
    - a) B's **effective value constraint** is **absent** or default.

- b) R's **effective value constraint** is fixed with the same string as B's.
- B. the must have an and the of the R's must be **valid** with respect to that wildcard, as defined in Wildcard allows Namespace Name For a value which is either a namespace name or absent to be valid with respect to a wildcard constraint (the value of a ) 1. 2. 3. .
3. For each attribute use in the of the whose is true, there must be an attribute use with an with the same and as its in the of the complex type definition itself whose is true.
  4. If there is an ,
    - A. The must also have one.
    - B. The complex type definition's 's must be a subset of the 's 's , as defined by Wildcard Subset For a namespace constraint (call it sub) to be an intensional subset of another namespace constraint (call it super) 1. 2. 3. .
    - C. Unless the is the , the complex type definition's 's must be identical to or stronger than the 's 's , where strict is stronger than lax is stronger than skip.
  5. A. The must be the .
    - B. i. The of the complex type definition must be a simple type definition
      - ii. a) The of the must be a simple type definition from which the is validly derived given the empty set as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. .
        - b) The must be mixed and have a particle which is **emptiable** as defined in Particle Emptiable For a particle to be emptiable 1. 2. .
    - C. i. The of the complex type itself must be empty
      - ii. a) The of the must also be empty.
      - b) The of the must be elementOnly or mixed and have a particle which is **emptiable** as defined in Particle Emptiable For a particle to be emptiable 1. 2. .
    - D. i. a) The of the complex type definition itself must be element-only
      - b) The of the complex type definition itself and of the must be mixed
    - ii. The particle of the complex type definition itself must be a **valid restriction** of the particle of the of the as defined in Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. .



Attempts to derive complex type definitions whose is element-only by restricting a whose is empty are not ruled out by this clause. However if the complex type definition itself has a non-pointless particle it will fail to satisfy Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. . On the other hand some type definitions with pointless element-only content, for example an empty , will satisfy Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. with respect to an empty , and so be valid restrictions.



If this constraint Derivation Valid (Restriction, Complex) If the is restriction 1. 2. 3. 4. 5. If this constraint holds of a complex type definition, it is a valid restriction of its . holds of a complex type definition, it is a *valid restriction* of its .



To restrict a complex type definition with a simple base type definition to empty, use a simple type definition with a fixed value of the empty string: this preserves the type information.

The following constraint defines a relation appealed to elsewhere in this specification.

### cos: Type Derivation OK (Complex)

For a complex type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction}

1. If B and D are not the same type definition, then the of D must not be in the subset.
2. A. B and D must be the same type definition.
  - B. B must be D's .
  - C. i. D's must not be the .
    - ii. a) D's is complex  
it must be validly derived from B given the subset as defined by this constraint.
    - b) D's is simple  
it must be validly derived from B given the subset as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. .



This constraint is used to check that when someone uses a type in a context where another type was expected (either via  `xsi:type`  or substitution groups), that the type used is actually derived from the expected type, and that that derivation does not involve a form of derivation which was ruled out by the expected type.



The wording of above appeals to a notion of component identity which is only incompletely defined by this version of this specification. In some cases, the wording of this specification does make clear the rules for component identity. These cases include:

- When they are both top-level components with the same component type, namespace name, and local name;
- When they are necessarily the same type definition (for example, when the two types definitions in question are the type definitions associated with two attribute or element declarations, which are discovered to be the same declaration);
- When they are the same by construction (for example, when an element's type definition defaults to being the same type definition as that of its substitution-group head or when a complex type definition inherits an attribute declaration from its base type definition).


In other cases two conforming implementations may disagree as to whether components are identical.

### 3.4.7. Built-in Complex Type Definition

There is a complex type definition nearly equivalent to the [ur-type definition](#) present in every schema by definition. It has the following properties:


Complex Type Definition of the Ur-Type anyType <http://www.w3.org/2001/XMLSchema> Itself restriction A pair consisting of mixed and a particle with the following properties: 1 1 a model group with the following properties: sequence a list containing one particle with the following properties: 0 unbounded a wildcard with the following properties: any lax The empty set a wildcard with the following properties:: any lax The empty set The empty set false

The mixed content specification together with the lax wildcard and attribute specification produce the defining property for the [ur-type definition](#), namely that *every* type definition is (eventually) a restriction of the [ur-type definition](#): its permissions and requirements are (nearly) the least restrictive possible.

 This specification does not provide an inventory of built-in complex type definitions for use in user schemas. A preliminary library of complex type definitions is available which includes both mathematical (e.g. `rational`) and utility (e.g. `array`) type definitions. In particular, there is a `text` type definition which is recommended for use as the type definition in element declarations intended for general text content, as it makes sensible provision for various aspects of internationalization. For more details, see the schema document for the type library at its namespace name: <http://www.w3.org/2001/03/XMLSchema/TypeLibrary.xsd>.

## 3.5. AttributeUses

An attribute use is a utility component which controls the occurrence and defaulting behavior of attribute declarations. It plays the same role for attribute declarations in complex types that particles play for element declarations.



```
<xs:complexType>
  . . .
  <xs:attribute ref="xml:lang" use="required"/>
  <xs:attribute ref="xml:space" default="preserve"/>
  <xs:attribute name="version" type="xs:number" fixed="1.0"/>
</xs:complexType>
```

XML representations which all involve attribute uses, illustrating some of the possibilities for controlling occurrence.

### 3.5.1. The Attribute Use Schema Component

The attribute use schema component has the following properties:

A boolean. An attribute declaration. Optional. A pair consisting of a value and one of default, fixed. determines whether this use of an attribute declaration requires an appropriate attribute information item to be present, or merely allows it.

provides the attribute declaration itself, which will in turn determine the simple type definition used.

allows for local specification of a default or fixed value. This must be consistent with that of the , in that if the specifies a fixed value, the only allowed is the same fixed value.

### 3.5.2. XML Representation of Attribute Use Components

Attribute uses correspond to all uses of which allow a `use` attribute. These in turn correspond to *two* components in each case, an attribute use and its (although note the latter is not new when the attribute use is a reference to a top-level attribute declaration). The appropriate mapping is described in § 3.2.2 – XML Representation of Attribute Declaration Schema Components on page 17.

### 3.5.3. Constraints on XML Representations of Attribute Uses

None as such.

### 3.5.4. Attribute Use Validation Rules

#### **cvc: Attribute Locally Valid (Use)**

For an attribute information item to be `valid` with respect to an attribute use its `normalized value` must match the canonical lexical representation of the attribute use's value, if it is present and fixed.

### 3.5.5. Attribute Use Information Set Contributions

None as such.

### 3.5.6. Constraints on Attribute Use Schema Components

All attribute uses (see § 3.5 – AttributeUses on page 55) must satisfy the following constraints.


#### **cos: Attribute Use Correct**

1. The values of the properties of an attribute use must be as described in the property tableau in § 3.5.1 – The Attribute Use Schema Component on page 55, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. If the has a fixed , then if the attribute use itself has a , it must also be fixed and its value must match that of the 's .

## 3.6. Attribute Group Definitions

A schema can name a group of attribute declarations so that they may be incorporated as a group into complex type definitions.

Attribute group definitions do not participate in `validation` as such, but the and of one or more complex type definitions may be constructed in whole or part by reference to an attribute group. Thus, attribute group definitions provide a replacement for some uses of XML's `parameter entity` facility. Attribute group definitions are provided primarily for reference from the XML representation of schema components (see and ).



```
<xs:attributeGroup name="myAttrGroup">
  <xs:attribute . . . />
  . . .
</xs:attributeGroup>
```

```

<xs:complexType name="myelement">
    . . .
    <xs:attributeGroup ref="myAttrGroup"/>
</xs:complexType>

```

XML representations for attribute group definitions. The effect is as if the attribute declarations in the group were present in the type definition.

### 3.6.1. The Attribute Group Definition Schema Component

The attribute group definition schema component has the following properties:

An NCName as defined by [XML-Namespaces]. Either [absent](#) or a namespace name, as defined in [XML-Namespaces]. A set of attribute uses. Optional. A wildcard. Optional. An annotation.

Attribute groups are identified by their and ; attribute group identities must be unique within an XML Schema. See § 4.2.3 – [References to schema components across namespaces](#) on page 110 for the use of component identifiers when importing one schema into another.

is a set attribute uses, allowing for local specification of occurrence and default or fixed values.

provides for an attribute wildcard to be included in an attribute group. See above under § 3.4 – [Complex Type Definitions](#) on page 38 for the interpretation of attribute wildcards during [validation](#).

See § 3.13 – [Annotations](#) on page 88 for information on the role of the property.

### 3.6.2. XML Representation of Attribute Group Definition Schema Components

The XML representation for an attribute group definition schema component is an element information item. It provides for naming a group of attribute declarations and an attribute wildcard for use by reference in the XML representation of complex type definitions and other attribute group definitions. The correspondences between the properties of the information item and properties of the component it corresponds to are as follows:

When an [attributeGroup](#) appears as a daughter of [complexType](#) or [attributeGroup](#), it corresponds to an attribute group definition as below. When it appears as a daughter of [complexType](#) or [attributeGroup](#), it does not correspond to any component as such.

The [actual value](#) of the [name](#) attribute The [actual value](#) of the [targetNamespace](#) attribute of the parent schema element information item. The union of the set of attribute uses corresponding to the children, if any, with the of the attribute groups to by the [actual values](#) of the [ref](#) attribute of the children, if any. As for the [complete wildcard](#) as described in § 3.4.2 – [XML Representation of Complex Type Definitions](#) on page 40. The annotation corresponding to the element information item in the children, if present, otherwise [absent](#).

The example above illustrates a pattern which recurs in the XML representation of schemas: The same element, in this case [attributeGroup](#), serves both to define and to incorporate by reference. In the first case the [name](#) attribute is required, in the second the [ref](#) attribute is required, and the element must be empty. These two are mutually exclusive, and also conditioned by context: the defining form, with a [name](#), must occur at the top level of a schema, whereas the referring form, with a [ref](#), must occur within a complex type definition or an attribute group definition.

### 3.6.3. Constraints on XML Representations of Attribute Group Definitions

#### src: Attribute Group Definition Representation OK

In addition to the conditions imposed on element information items by the schema for schemas,

1. The corresponding attribute group definition, if any, must satisfy the conditions set out in § 3.6.6 – Constraints on Attribute Group Definition Schema Components on page 58.
2. If or in the correspondence specification in § 3.4.2 – XML Representation of Complex Type Definitions on page 40 for , as referenced above, is satisfied, the intensional intersection must be expressible, as defined in Attribute Wildcard Intersection For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2): 1. 2. 3. 4. 5. 6. In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required. .
3. Circular group reference is disallowed outside . That is, unless this element information item's parent is , then among the children, if any, there must not be an with `ref` attribute which resolves to the component corresponding to this . Indirect circularity is also ruled out. That is, when QName resolution (Schema Document) For a QName to resolve to a schema component of a specified kind 1. 2. 3. 4. is applied to a QName arising from any s with a `ref` attribute among the children, it must not be the case that a QName is encountered at any depth which resolves to the component corresponding to this .

### 3.6.4. Attribute Group Definition Validation Rules

None as such.

### 3.6.5. Attribute Group Definition Information Set Contributions

None as such.

### 3.6.6. Constraints on Attribute Group Definition Schema Components

All attribute group definitions (see § 3.6 – Attribute Group Definitions on page 56) must satisfy the following constraint.

#### cos: Attribute Group Definition Properties Correct

1. The values of the properties of an attribute group definition must be as described in the property tableau in § 3.6.1 – The Attribute Group Definition Schema Component on page 57, modulo the impact of § 5.3 – Missing Sub-components on page 117;
2. Two distinct members of the must not have s both of whose s match and whose s are identical.
3. Two distinct members of the must not have s both of whose s are or are derived from ID.

## 3.7. Model Group Definitions

A model group definition associates a name and optional annotations with a § 2.2.3.1 – Model Group on page 8. By reference to the name, the entire model group can be incorporated by reference into a .

Model group definitions are provided primarily for reference from the § 3.4.2 – XML Representation of Complex Type Definitions on page 40 (see and ). Thus, model group definitions provide a replacement for some uses of XML's [parameter entity](#) facility.

```

☞ <xs:group name="myModelGroup">
  <xs:sequence>
    <xs:element ref="something"/>
    . . .
  </xs:sequence>
</xs:group>

<xs:complexType name="trivial">
  <xs:group ref="myModelGroup"/>
  <xs:attribute ../>
</xs:complexType>

<xs:complexType name="moreSo">
  <xs:choice>
    <xs:element ref="anotherThing"/>
    <xs:group ref="myModelGroup"/>
  </xs:choice>
  <xs:attribute ../>
</xs:complexType>

```

A minimal model group is defined and used by reference, first as the whole content model, then as one alternative in a choice.

### 3.7.1. The Model Group Definition Schema Component

The model group definition schema component has the following properties:

An NCName as defined by [XML-Namespaces]. Either **absent** or a namespace name, as defined in [XML-Namespaces]. A model group. Optional. An annotation.

Model group definitions are identified by their and ; model group identities must be unique within an XML Schema. See § 4.2.3 – References to schema components across namespaces on page 110 for the use of component identifiers when importing one schema into another.

Model group definitions *per se* do not participate in **validation**, but the of a particle may correspond in whole or in part to a model group from a model group definition.

is the § 2.2.3.1 – Model Group on page 8 for which the model group definition provides a name.

See § 3.13 – Annotations on page 88 for information on the role of the property.

### 3.7.2. XML Representation of Model Group Definition Schema Components

The XML representation for a model group definition schema component is a element information item. It provides for naming a model group for use by reference in the XML representation of complex type definitions and model groups. The correspondences between the properties of the information item and properties of the component it corresponds to are as follows:

If there is a name attribute (in which case the item will have or as parent), then the item corresponds to a model group definition component with properties as follows:

The **actual value** of the name attribute The **actual value** of the targetNamespace attribute of the parent schema element information item. A model group which is the of a particle corresponding to the , or among the children (there must be one). The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

Otherwise, the item will have a `ref` attribute, in which case it corresponds to a particle component with properties as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

The **actual value** of the `minOccurs` attribute, if present, otherwise 1. unbounded, if the `maxOccurs` attribute equals unbounded, otherwise the **actual value** of the `maxOccurs` attribute, if present, otherwise 1. The of the model group definition to by the **actual value** of the `ref` attribute

The name of this section is slightly misleading, in that the second, un-named, case above (with a `ref` and no name) is not really a named model group at all, but a reference to one. Also note that in the first (named) case above no reference is made to `minOccurs` or `maxOccurs`: this is because the schema for schemas does not allow them on the child of when it is named. This in turn is because the and of the particles which *refer* to the definition are what count.

Given the constraints on its appearance in content models, an should only occur as the only item in the children of a named model group definition or a content model: see § 3.8.6 – Constraints on Model Group Schema Components on page 63.

### 3.7.3. Constraints on XML Representations of Model Group Definitions

#### **src: Model Group Definition Representation OK**

In addition to the conditions imposed on element information items by the schema for schemas, the corresponding model group definition, if any, must satisfy the conditions set out in § 3.8.6 – Constraints on Model Group Schema Components on page 63.

### 3.7.4. Model Group Definition Validation Rules

None as such.

### 3.7.5. Model Group Definition Information Set Contributions

None as such.

### 3.7.6. Constraints on Model Group Definition Schema Components

All model group definitions (see § 3.7 – Model Group Definitions on page 58) must satisfy the following constraint.

#### **cos: Model Group Definition Properties Correct**

The values of the properties of a model group definition must be as described in the property tableau in § 3.7.1 – The Model Group Definition Schema Component on page 59, modulo the impact of § 5.3 – Missing Sub-components on page 117.

## 3.8. Model Groups

When the children of element information items are not constrained to be empty or by reference to a simple type definition (§ 3.14 – Simple Type Definitions on page 89), the sequence of element information item children content may be specified in more detail with a model group. Because the property of a particle can be a model group, and model groups contain particles, model groups can indirectly contain other model groups; the grammar for content models is therefore recursive.



```

☞ <xs:all>
  <xs:element ref="cats"/>
  <xs:element ref="dogs"/>
</xs:all>

<xs:sequence>
  <xs:choice>
    <xs:element ref="left"/>
    <xs:element ref="right"/>
  </xs:choice>
  <xs:element ref="landmark"/>
</xs:sequence>

```

XML representations for the three kinds of model group, the third nested inside the second.

### 3.8.1. The Model Group Schema Component

The model group schema component has the following properties:

One of all, choice or sequence. A list of particles Optional. An annotation. specifies a sequential (sequence), disjunctive (choice) or conjunctive (all) interpretation of the . This in turn determines whether the element information item children **validated** by the model group must:

- (sequence) correspond, in order, to the specified ;
- (choice) corresponded to exactly one of the specified ;
- (all) contain all and only exactly zero or one of each element specified in . The elements can occur in any order. In this case, to reduce implementation complexity, is restricted to contain local and top-level element declarations only, with =0 or 1, =1.

When two or more particles contained directly or indirectly in the of a model group have identically named element declarations as their , the type definitions of those declarations must be the same. By 'indirectly' is meant particles within the of a group which is itself the of a directly contained particle, and so on recursively.

See § 3.13 – Annotations on page 88 for information on the role of the property.

### 3.8.2. XML Representation of Model Group Schema Components

The XML representation for a model group schema component is either an , a or a element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

Each of the above items corresponds to a particle containing a model group, with properties as follows (unless minOccurs=maxOccurs=0, in which case the item corresponds to no component at all):

The **actual value** of the minOccurs attribute, if present, otherwise 1. unbounded, if the maxOccurs attribute equals unbounded, otherwise the **actual value** of the maxOccurs attribute, if present, otherwise 1. A model group as given below: One of all, choice, sequence depending on the element information item. A sequence of particles corresponding to all the , , , or items among the children, in order. The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

### 3.8.3. Constraints on XML Representations of Model Groups

#### src: Model Group Representation OK

In addition to the conditions imposed on , and element information items by the schema for schemas, the corresponding particle and model group must satisfy the conditions set out in § 3.8.6 – Constraints on Model Group Schema Components on page 63 and § 3.9.6 – Constraints on Particle Schema Components on page 68.

### 3.8.4. Model Group Validation Rules

#### cvc: Element Sequence Valid

Define a *partition* of a sequence as a sequence of sub-sequences, some or all of which may be empty, such that concatenating all the sub-sequences yields the original sequence.

For a sequence (possibly empty) of element information items to be locally **valid** with respect to a model group

1. the is sequence

there must be a **partition** of the sequence into  $n$  sub-sequences where  $n$  is the length of such that each of the sub-sequences in order is **valid** with respect to the corresponding particle in the as defined in Element Sequence Locally Valid (Particle) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle 1. 2. 3. Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. .

2. the is choice

there must be a particle among the such that the sequence is **valid** with respect to that particle as defined in Element Sequence Locally Valid (Particle) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle 1. 2. 3. Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. .

3. the is all

there must be a **partition** of the sequence into  $n$  sub-sequences where  $n$  is the length of such that there is a one-to-one mapping between the sub-sequences and the where each sub-sequence is **valid** with respect to the corresponding particle as defined in Element Sequence Locally Valid (Particle) For a sequence (possibly empty) of element information items to be locally valid with respect to a particle 1. 2. 3. Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is not validatable by a wildcard which accepts the head's namespace but not its own. .

Nothing in the above should be understood as ruling out groups whose is empty: although no sequence can be **valid** with respect to such a group whose is choice, the empty sequence *is* **valid** with respect to empty groups whose is sequence or all.



The above definition is implicitly non-deterministic, and should not be taken as a recipe for implementations. Note in particular that when is all, particles is restricted to a list of local and top-level element declarations (see § 3.8.6 – Constraints on Model Group Schema Components on page 63). A much simpler implementation is possible than

would arise from a literal interpretation of the definition above; informally, the content is **valid** when each declared element occurs exactly once (or at most once, if is 0), and each is **valid** with respect to its corresponding declaration. The elements can occur in arbitrary order.

### 3.8.5. Model Group Information Set Contributions

None as such.

### 3.8.6. Constraints on Model Group Schema Components

All model groups (see § 3.8 – Model Groups on page 60) must satisfy the following constraints.

#### cos: Model Group Correct

1. The values of the properties of a model group must be as described in the property tableau in § 3.8.1 – The Model Group Schema Component on page 61, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. Circular groups are disallowed. That is, within the of a group there must not be at any depth a particle whose is the group itself.

#### cos: All Group Limited

When a model group has all, then

1. It appears only as the value of one or both of the following properties:
  - A. the property of a model group definition.
  - B. the property of a particle with =1 which is part of a pair which constitutes the of a complex type definition.
2. The of all the particles in the of the group must be 0 or 1.

#### cos: Element Declarations Consistent

If the contains, either directly, indirectly (that is, within the of a contained model group, recursively) or **implicitly** two or more element declaration particles with the same and , then all their type definitions must be the same top-level definition, that is,

1. all their s must have a non-**absent** .
2. all their s must have the same .
3. all their s must have the same .

A list of particles *implicitly contains* an element declaration if a member of the list contains that element declaration in its **substitution group**.

#### cos: Unique Particle Attribution

A content model must be formed such that during **validation** of an element information item sequence, the particle component contained directly, indirectly or **implicitly** therein with which to attempt to **validate**

each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence.



This constraint reconstructs for XML Schema the equivalent constraints of [XML 1.0 (Second Edition)] and SGML. Given the presence of element substitution groups and wildcards, the concise expression of this constraint is difficult, see [Appendix H – Analysis of the Unique Particle Attribution Constraint \(non-normative\)](#) on page 121 for further discussion.

Since this constraint is expressed at the component level, it applies to content models whose origins (e.g. via type derivation and references to named model groups) are no longer evident. So particles at different points in the content model are always distinct from one another, even if they originated from the same named model group.



Because locally-scoped element declarations may or may not have a , the scope of declarations is *not* relevant to enforcing either of the two preceding constraints.

The following constraints define relations appealed to elsewhere in this specification.

#### **cos: Effective Total Range (all and sequence)**

The effective total range of a particle whose is a group whose is all or sequence is a pair of minimum and maximum, as follows:

##### ***minimum***

The product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ).

##### ***maximum***

unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded, or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ).

#### **cos: Effective Total Range (choice)**

The effective total range of a particle whose is a group whose is choice is a pair of minimum and maximum, as follows:

##### ***minimum***

The product of the particle's and the minimum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ).

##### ***maximum***

unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded, or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the maximum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ).

### 3.9. Particles

As described in § 3.8 – Model Groups on page 60, particles contribute to the definition of content models.

```

☞ <xs:element ref="egg" minOccurs="12" maxOccurs="12"/>

<xs:group ref="omelette" minOccurs="0"/>

<xs:any maxOccurs="unbounded"/>
    
```

XML representations which all involve particles, illustrating some of the possibilities for controlling occurrence.

#### 3.9.1. The Particle Schema Component

The particle schema component has the following properties:

A non-negative integer. Either a non-negative integer or unbounded. One of a model group, a wildcard, or an element declaration.

In general, multiple element information item children, possibly with intervening character children if the content type is mixed, can be **validated** with respect to a single particle. When the is an element declaration or wildcard, determines the minimum number of such element children that can occur. The number of such children must be greater than or equal to . If is 0, then occurrence of such children is optional.

Again, when the is an element declaration or wildcard, the number of such element children must be less than or equal to any numeric specification of ; if is unbounded, then there is no upper bound on the number of such children.

When the is a model group, the permitted occurrence range is determined by a combination of and and the occurrence ranges of the 's .

#### 3.9.2. XML Representation of Particle Components

Particles correspond to all three elements ( not immediately within , not immediately within and ) which allow `minOccurs` and `maxOccurs` attributes. These in turn correspond to *two* components in each case, a particle and its . The appropriate mapping is described in § 3.3.2 – XML Representation of Element Declaration Schema Components on page 24, § 3.8.2 – XML Representation of Model Group Schema Components on page 61 and § 3.10.2 – XML Representation of Wildcard Schema Components on page 75 respectively.

#### 3.9.3. Constraints on XML Representations of Particles

None as such.

#### 3.9.4. Particle Validation Rules

##### **cvc: Element Sequence Locally Valid (Particle)**

For a sequence (possibly empty) of element information items to be locally **valid** with respect to a particle

1. the is a wildcard
  - A. The length of the sequence must be greater than or equal to the .
  - B. If is a number, the length of the sequence must be less than or equal to the .

C. Each element information item in the sequence must be **valid** with respect to the wildcard as defined by Item Valid (Wildcard) For an element or attribute information item to be locally valid with respect to a wildcard constraint its namespace name must be valid with respect to the wildcard constraint, as defined in . When this constraint applies 1. 2. 3. .

2. the is an element declaration

A. The length of the sequence must be greater than or equal to the .

B. If is a number, the length of the sequence must be less than or equal to the .

C. For each element information item in the sequence

i. The element declaration is local (i.e. its must not be global), its is false, the element information item's namespace name is identical to the element declaration's (where an **absent** is taken to be identical to a namespace name with no value) and the element information item's local name matches the element declaration's .

In this case the element declaration is the **context-determined declaration** for the element information item with respect to Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . and Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. .


ii. The element declaration is top-level (i.e. its is global), is false, the element information item's namespace name is identical to the element declaration's (where an **absent** is taken to be identical to a namespace name with no value) and the element information item's local name matches the element declaration's .

In this case the element declaration is the **context-determined declaration** for the element information item with respect to Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . and Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. .

iii. The element declaration is top-level (i.e. its is global), its does not contain substitution, the local and namespace name of the element information item resolve to an element declaration, as defined in QName resolution (Instance) A pair of a local name and a namespace name (or absent) resolve to a schema component of a specified kind in the context of validation by appeal to the appropriate property of the schema being used for the assessment. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, 1. 2. 3. 4. 5. 6. The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair. -- call this declaration the *substituting declaration* and the *substituting declaration* together with the particle's element declaration's is validly substitutable for the particle's element declaration as defined in Substitution Group OK (Transitive) For an element declaration (call it D) to be validly substitutable for another element declaration (call it C) subject to a blocking constraint (a subset of {substitution, extension, restriction}), the value of a ) 1. 2. .

In this case the *substituting declaration* is the *context-determined declaration* for the element information item with respect to Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . and Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. .

- 3. the is a model group
  - A. There is a *partition* of the sequence into n sub-sequences such that n is greater than or equal to .
  - B. If is a number, n must be less than or equal to .
  - C. Each sub-sequence in the *partition* is *valid* with respect to that model group as defined in Element Sequence Valid Define a partition of a sequence as a sequence of sub-sequences, some or all of which may be empty, such that concatenating all the sub-sequences yields the original sequence. For a sequence (possibly empty) of element information items to be locally valid with respect to a model group 1. 2. 3. Nothing in the above should be understood as ruling out groups whose is empty: although no sequence can be valid with respect to such a group whose is choice, the empty sequence is valid with respect to empty groups whose is sequence or all. .

 Clauses and do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is *not* validatable by a wildcard which accepts the head's namespace but not its own.



### 3.9.5. Particle Information Set Contributions

None as such.

### 3.9.6. Constraints on Particle Schema Components

All particles (see § 3.9 – Particles on page 65) must satisfy the following constraints.

#### cos: Particle Correct

1. The values of the properties of a particle must be as described in the property tableau in § 3.9.1 – The Particle Schema Component on page 65, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. If is not unbounded, that is, it has a numeric value, then
  - A. must not be greater than .
  - B. must be greater than or equal to 1.

The following constraints define relations appealed to elsewhere in this specification.

#### cos: Particle Valid (Extension)

For a particle (call it E, for extension) to be a *valid extension* of another particle (call it B, for base)

1. They are the same particle.
2. E's ==1 and its is a sequence group whose ' first member is a particle all of whose properties, recursively, are identical to those of B, with the exception of annotation properties.

The approach to defining a type by restricting another type definition set out here is designed to ensure that types defined in this way are guaranteed to be a subset of the type they restrict. This is accomplished by requiring a clear mapping between the components of the base type definition and the restricting type definition. Permissible mappings are set out below via a set of recursive definitions, bottoming out in the obvious cases, e.g. where an (restricted) element declaration corresponds to another (base) element declaration with the same name and type but the same or wider range of occurrence.



The structural correspondence approach to guaranteeing the subset relation set out here is necessarily verbose, but has the advantage of being checkable in a straightforward way. The working group solicits feedback on how difficult this is in practice, and on whether other approaches are found to be viable.

#### cos: Particle Valid (Restriction)

For a particle (call it R, for restriction) to be a *valid restriction* of another particle (call it B, for base)

1. They are the same particle.
2. depending on the kind of particle, per the table below, with the qualifications that
  - A. Any top-level element declaration particle (in R or B) which is the of one or more other element declarations and whose [substitution group](#) contains at least one element declaration other than itself is treated as if it were a choice group whose and are those of the particle, and whose consists of one particle with and of 1 for each of the declarations in its [substitution group](#).

B. Any pointless occurrences of , or are ignored, where pointlessness is understood as follows:

- i. is empty.
- ii. a) The particle within which this appears has and of 1.
  - b) 1) The 's has only one member.
  - 2) The particle within which this appears is itself among the of a .
- i. is empty.
- ii. has only one member.
- i. is empty and the particle within which this appears has of 0.
- ii. a) The particle within which this appears has and of 1.
  - b) 1) The 's has only one member.
  - 2) The particle within which this appears is itself among the of a .

NameAnd- TypeOK NSCompat Recurse- AsIfGroup Recurse- AsIfGroup RecurseAs- IfGroup  
 NSSubset Forbidden Forbidden Forbidden Forbidden NSRecurse- CheckCardinality Recurse Forbidden  
 Forbidden Forbidden NSRecurse- CheckCardinality RecurseLax Forbidden Forbidden Forbidden  
 NSRecurse- CheckCardinality Recurse- Unordered MapAndSum Recurse Forbidden

**cos: Occurrence Range OK**

For a particle's occurrence range to be a valid restriction of another's occurrence range

1. Its is greater than or equal to the other's .
2. A. The other's is unbounded.
  - B. Both are numbers, and the particle's is less than or equal to the other's.

**cos: Particle Restriction OK (Elt:Elt -- NameAndTypeOK)**

For an element declaration particle to be a **valid restriction** of another element declaration particle

1. The declarations' s and s are the same.
2. R's occurrence range is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .
3. A. Both B's declaration's and R's declaration's are global.
  - B. i. Either B's is true or R's is false.
    - ii. either B's declaration's is absent, or is not fixed, or R's declaration's is fixed with the same value.
    - iii. R's declaration's is a subset of B's declaration's , if any.
    - iv. R's declaration's is a superset of B's declaration's .

- v. R's is validly derived given {extension, list, union} from B's as defined by Type Derivation OK (Complex) For a complex type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction} 1. 2. or Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. , as appropriate.



The above constraint on means that in deriving a type by restriction, any contained type definitions must themselves be explicitly derived by restriction from the corresponding type definitions in the base definition, or be one of the member types of a corresponding union..

#### **cos: Particle Derivation OK (Elt:Any -- NSCompat)**

For an element declaration particle to be a [valid restriction](#) of a wildcard particle

1. The element declaration's is [valid](#) with respect to the wildcard's as defined by Wildcard allows Namespace Name For a value which is either a namespace name or absent to be valid with respect to a wildcard constraint (the value of a ) 1. 2. 3. .
2. R's occurrence range is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .


#### **cos: Particle Derivation OK (Elt:All/Choice/Sequence -- RecurseAsIfGroup)**

For an element declaration particle to be a [valid restriction](#) of a group particle (all, choice or sequence) a group particle of the variety corresponding to B's, with and of 1 and with consisting of a single particle the same as the element declaration must be a [valid restriction](#) of the group as defined by Particle Derivation OK (All:All,Sequence:Sequence -- Recurse) For an all or sequence group particle to be a valid restriction of another group particle with the same 1. 2. Although the validation semantics of an all group does not depend on the order of its particles, derived all groups are required to match the order of their base in order to simplify checking that the derivation is OK. A complete functional mapping is order-preserving if each particle r in the domain R maps to a particle b in the range B which follows (not necessarily immediately) the particle in the range B mapped to by the predecessor of r, if any, where predecessor and follows are defined with respect to the order of the lists which constitute R and B. , Particle Derivation OK (Choice:Choice -- RecurseLax) For a choice group particle to be a valid restriction of another choice group particle 1. 2. Although the validation semantics of a choice group does not depend on the order of its particles, derived choice groups are required to match the order of their base in order to simplify checking that the derivation is OK. or Particle Derivation OK (All:All,Sequence:Sequence -- Recurse) For an all or sequence group particle to be a valid restriction of another group particle with the same 1. 2. Although the validation semantics of an all group does not depend on the order of its particles, derived all groups are required to match the order of their base in order to simplify checking that the derivation is OK. A complete functional mapping is order-preserving if each particle r in the domain R maps to a particle b in the range B which follows (not necessarily immediately) the particle in the range B mapped to by the predecessor of r, if any, where predecessor and follows are defined with respect to the order of the lists which constitute R and B. , depending on whether the group is all, choice or sequence.

**cos: Particle Derivation OK (Any:Any -- NSSubset)**

For a wildcard particle to be a [valid restriction](#) of another wildcard particle

1. R's occurrence range must be a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .
2. R's must be an intensional subset of B's as defined by Wildcard Subset For a namespace constraint (call it sub) to be an intensional subset of another namespace constraint (call it super) 1. 2. 3. .
3. Unless B is the content model wildcard of the , R's must be identical to or stronger than B's , where strict is stronger than lax is stronger than skip.

 The exception to the third clause above for derivations from the is necessary as its wildcards have a of lax, so without this exception, no use of wildcards with of skip would be possible.

**cos: Particle Derivation OK (All/Choice/Sequence:Any -- NSRecurseCheckCardinality)**

For a group particle to be a [valid restriction](#) of a wildcard particle

1. Every member of the of the group is a [valid restriction](#) of the wildcard as defined by Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. .
2. The effective total range of the group, as defined by Effective Total Range (all and sequence) The effective total range of a particle whose is a group whose is all or sequence is a pair of minimum and maximum, as follows: minimum The product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). maximum unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded, or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). (if the group is all or sequence) or Effective Total Range (choice) The effective total range of a particle whose is a group whose is choice is a pair of minimum and maximum, as follows: minimum The product of the particle's and the minimum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). maximum unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded, or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the maximum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). (if it is choice) is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .

**cos: Particle Derivation OK (All:All,Sequence:Sequence -- Recurse)**

For an all or sequence group particle to be a [valid restriction](#) of another group particle with the same

1. R's occurrence range is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .
2. There is a complete **order-preserving** functional mapping from the particles in the of R to the particles in the of B such that
  - A. Each particle in the of R is a **valid restriction** of the particle in the of B it maps to as defined by Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. .
  - B. All particles in the of B which are not mapped to by any particle in the of R are **emptiable** as defined by Particle Emptiable For a particle to be emptiable 1. 2. .



Although the **validation** semantics of an all group does not depend on the order of its particles, derived all groups are required to match the order of their base in order to simplify checking that the derivation is OK.

A complete functional mapping is *order-preserving* if each particle *r* in the domain *R* maps to a particle *b* in the range *B* which follows (not necessarily immediately) the particle in the range *B* mapped to by the predecessor of *r*, if any, where “predecessor” and “follows” are defined with respect to the order of the lists which constitute *R* and *B*.

#### **cos: Particle Derivation OK (Choice:Choice -- RecurseLax)**

For a choice group particle to be a **valid restriction** of another choice group particle

1. R's occurrence range is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. ;
2. There is a complete **order-preserving** functional mapping from the particles in the of R to the particles in the of B such that each particle in the of R is a **valid restriction** of the particle in the of B it maps to as defined by Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. .




Although the **validation** semantics of a choice group does not depend on the order of its particles, derived choice groups are required to match the order of their base in order to simplify checking that the derivation is OK.

#### **cos: Particle Derivation OK (Sequence:All -- RecurseUnordered)**

For a sequence group particle to be a **valid restriction** of an all group particle

1. R's occurrence range is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .
2. There is a complete functional mapping from the particles in the of R to the particles in the of B such that
  - A. No particle in the of B is mapped to by more than one of the particles in the of R;
  - B. Each particle in the of R is a **valid restriction** of the particle in the of B it maps to as defined by Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. ;


- C. All particles in the of B which are not mapped to by any particle in the of R are **emptiable** as defined by Particle Emptiable For a particle to be emptiable 1. 2. .


 Although this clause allows reordering, because of the limits on the contents of all groups the checking process can still be deterministic.

**cos: Particle Derivation OK (Sequence:Choice -- MapAndSum)**

For a sequence group particle to be a **valid restriction** of a choice group particle

1. There is a complete functional mapping from the particles in the of R to the particles in the of B such that each particle in the of R is a **valid restriction** of the particle in the of B it maps to as defined by Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2. .
2. The pair consisting of the product of the of R and the length of its and unbounded if is unbounded otherwise the product of the of R and the length of its is a valid restriction of B's occurrence range as defined by Occurrence Range OK For a particle's occurrence range to be a valid restriction of another's occurrence range 1. 2. .

 This clause is in principle more restrictive than absolutely necessary, but in practice will cover all the likely cases, and is much easier to specify than the fully general version.

 This case allows the “unfolding” of iterated disjunctions into sequences. It may be particularly useful when the disjunction is an implicit one arising from the use of substitution groups.

**cos: Particle Emptiable**

For a particle to be *emptiable*

1. Its is 0.
2. Its is a group and the minimum part of the effective total range of that group, as defined by Effective Total Range (all and sequence) The effective total range of a particle whose is a group whose is all or sequence is a pair of minimum and maximum, as follows: minimum The product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). maximum unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded, or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the sum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). (if the group is all or sequence) or Effective Total Range (choice) The effective total range of a particle whose is a group whose is choice is a pair of minimum and maximum, as follows: minimum The product of the particle's and the minimum of the of every wildcard or element declaration particle in the group's and the minimum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). maximum unbounded if the of any wildcard or element declaration particle in the group's or the maximum part of the effective total range of any of the group particles in the group's is unbounded,

or if any of those is non-zero and the of the particle itself is unbounded, otherwise the product of the particle's and the maximum of the of every wildcard or element declaration particle in the group's and the maximum part of the effective total range of each of the group particles in the group's (or 0 if there are no ). (if it is choice), is 0.

## 3.10. Wildcards

In order to exploit the full potential for extensibility offered by XML plus namespaces, more provision is needed than DTDs allow for targeted flexibility in content models and attribute declarations. A wildcard provides for [validation](#) of attribute and element information items dependent on their namespace name, but independently of their local name.



```
<xs:any processContents="skip"/>
```

```
<xs:any namespace="##other" processContents="lax"/>
```

```
<xs:any namespace="http://www.w3.org/1999/XSL/Transform"/>
```

```
<xs:any namespace="##targetNamespace"/>
```

```
<xs:anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
```

XML representations of the four basic types of wildcard, plus one attribute wildcard.

### 3.10.1. The Wildcard Schema Component

The wildcard schema component has the following properties:

One of any; a pair of not and a namespace name or [absent](#); or a set whose members are either namespace names or [absent](#). One of skip, lax or strict. Optional. An annotation. provides for [validation](#) of attribute and element items that:

1. (any) have any namespace or are not namespace-qualified;
2. (not and a namespace name) are namespace-qualified with a namespace other than the specified namespace name;
3. (not and [absent](#)) are namespace-qualified;
4. (a set whose members are either namespace names or [absent](#)) have any of the specified namespaces and/or, if [absent](#) is included in the set, are unqualified.

controls the impact on [assessment](#) of the information items allowed by wildcards, as follows:

#### *strict*

There must be a top-level declaration for the item available, or the item must have an `xsi:type`, and the item must be [valid](#) as appropriate.

#### *skip*

No constraints at all: the item must simply be well-formed XML.



*lax*

If the item has a uniquely determined declaration available, it must be [valid](#) with respect to that definition, that is, [validate](#) if you can, don't worry if you can't.

See [§ 3.13 – Annotations](#) on page 88 for information on the role of the property.

### 3.10.2. XML Representation of Wildcard Schema Components

The XML representation for a wildcard schema component is an or element information item. The correspondences between the properties of an information item and properties of the components it corresponds to are as follows (see and for the correspondences for ):

A particle containing a wildcard, with properties as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

The [actual value](#) of the `minOccurs` attribute, if present, otherwise 1. unbounded, if the `maxOccurs` attribute equals unbounded, otherwise the [actual value](#) of the `maxOccurs` attribute, if present, otherwise 1. A wildcard as given below: Dependent on the [actual value](#) of the `namespace` attribute: if absent, then any, otherwise as follows:

**##any**

any

**##other**

a pair of not and the [actual value](#) of the `targetNamespace` attribute of the ancestor element information item if present, otherwise [absent](#).

**otherwise**

a set whose members are namespace names corresponding to the space-delimited substrings of the string, except

1. if one such substring is `##targetNamespace`, the corresponding member is the [actual value](#) of the `targetNamespace` attribute of the ancestor element information item if present, otherwise [absent](#).
2. if one such substring is `##local`, the corresponding member is [absent](#).

The [actual value](#) of the `processContents` attribute, if present, otherwise strict. The annotation corresponding to the element information item in the children, if present, otherwise [absent](#).

Wildcards are subject to the same ambiguity constraints ( Unique Particle Attribution A content model must be formed such that during validation of an element information item sequence, the particle component contained directly, indirectly or implicitly therein with which to attempt to validate each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence. This constraint reconstructs for XML Schema the equivalent constraints of and SGML. Given the presence of element substitution groups and wildcards, the concise expression of this constraint is difficult, see for further discussion. Since this constraint is expressed at the component level, it applies to content models whose origins (e.g. via type derivation and references to named model groups) are no longer evident. So particles at different points in the content model are always distinct from one another, even if they originated from the same named model group. ) as other content model particles: If an instance element could match either an explicit particle and a wildcard, or one of two wildcards, within the content model of a type, that model is in error.

### 3.10.3. Constraints on XML Representations of Wildcards

#### src: Wildcard Representation OK

In addition to the conditions imposed on element information items by the schema for schemas, the corresponding particle and model group must satisfy the conditions set out in § 3.8.6 – Constraints on Model Group Schema Components on page 63 and § 3.9.6 – Constraints on Particle Schema Components on page 68.

### 3.10.4. Wildcard Validation Rules

#### cvc: Item Valid (Wildcard)

For an element or attribute information item to be locally **valid** with respect to a wildcard constraint its namespace name must be **valid** with respect to the wildcard constraint, as defined in Wildcard allows Namespace Name For a value which is either a namespace name or absent to be valid with respect to a wildcard constraint (the value of a ) 1. 2. 3. .

When this constraint applies

1. is lax

the item has no **context-determined declaration** with respect to Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation info set it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. , Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . and Schema-Validity Assessment (Attribute) The schema-validity assessment of an attribute information item depends on its validation alone. During validation, associations between element and attribute information items among the children and attributes on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the context-determined declarations. See (in ) for attribute declarations, (in ) for element declarations. For an attribute information item's schema-validity to have been assessed 1. 2. 3. For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been strictly assessed. .

2. is strict

the item's **context-determined declaration** is mustFind.

3. is skip

the item's **context-determined declaration** is skip.

**cvc: Wildcard allows Namespace Name**

For a value which is either a namespace name or [absent](#) to be [valid](#) with respect to a wildcard constraint (the value of a )

1. The constraint must be any.
2. A. The constraint is a pair of not and a namespace name or [absent](#) (call this the *namespace test*).  
B. The value must not be identical to the [namespace test](#).  
C. The value must not be [absent](#).
3. The constraint is a set, and the value is identical to one of the members of the set.

**3.10.5. Wildcard Information Set Contributions**

None as such.

**3.10.6. Constraints on Wildcard Schema Components**

All wildcards (see § 3.10 – [Wildcards](#) on page 74) must satisfy the following constraint.

**cos: Wildcard Properties Correct**

The values of the properties of a wildcard must be as described in the property tableau in § 3.10.1 – [The Wildcard Schema Component](#) on page 74, modulo the impact of § 5.3 – [Missing Sub-components](#) on page 117.

The following constraints define a relation appealed to elsewhere in this specification.

**cos: Wildcard Subset**

For a namespace constraint (call it sub) to be an intensional subset of another namespace constraint (call it super)

1. super must be any.
2. A. sub must be a pair of not and a value (a namespace name or [absent](#)).  
B. super must be a pair of not and the same value.
3. A. sub must be a set whose members are either namespace names or [absent](#).  
B. i. super must be the same set or a superset thereof.  
ii. super must be a pair of not and a value (a namespace name or [absent](#)) and neither that value nor [absent](#) must be in sub's set.

**cos: Attribute Wildcard Union**

For a wildcard's value to be the intensional union of two other such values (call them O1 and O2):

1. O1 and O2 are the same value  
that value must be the value.

2. either O1 or O2 is any  
any must be the value.
3. both O1 and O2 are sets of (namespace names or **absent**)  
the union of those sets must be the value.
4. the two are negations of different values (namespace names or **absent**)  
a pair of not and **absent** must be the value.
5. either O1 or O2 is a pair of not and a namespace name and the other is a set of (namespace names or **absent**) (call this set S)
  - A. the set S includes both the negated namespace name and **absent**  
any must be the value.
  - B. the set S includes the negated namespace name but not **absent**  
a pair of not and **absent** must be the value.
  - C. the set S includes **absent** but not the negated namespace name  
the union is not expressible.
  - D. the set S does not include either the negated namespace name or **absent**  
whichever of O1 or O2 is a pair of not and a namespace name must be the value.
6. either O1 or O2 is a pair of not and **absent** and the other is a set of (namespace names or **absent**) (again, call this set S)
  - A. the set S includes **absent**  
any must be the value.
  - B. the set S does not include **absent**  
a pair of not and **absent** must be the value.

In the case where there are more than two values, the intensional union is determined by identifying the intensional union of two of the values as above, then the intensional union of that value with the third (providing the first union was expressible), and so on as required.

### **cos: Attribute Wildcard Intersection**

For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2):

1. O1 and O2 are the same value  
that value must be the value.
2. either O1 or O2 is any  
the other must be the value.
3. either O1 or O2 is a pair of not and a value (a namespace name or **absent**) and the other is a set of (namespace names or **absent**)

that set, minus the negated value if it was in the set, minus [absent](#) if it was in the set, must be the value.


4. both O1 and O2 are sets of (namespace names or [absent](#))  
the intersection of those sets must be the value.
5. the two are negations of different namespace names  
the intersection is not expressible.
6. the one is a negation of a namespace name and the other is a negation of [absent](#)  
the one which is the negation of a namespace name must be the value.

In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required.

### 3.11. Identity-constraint Definitions

Identity-constraint definition components provide for uniqueness and reference constraints with respect to the contents of multiple elements and attributes.

```

 <xs:key name="fullName">
  <xs:selector xpath="//person"/>
  <xs:field xpath="forename"/>
  <xs:field xpath="surname"/>
</xs:key>

<xs:keyref name="personRef" refer="fullName">
  <xs:selector xpath="//personPointer"/>
  <xs:field xpath="@first"/>
  <xs:field xpath="@last"/>
</xs:keyref>

<xs:unique name="nearlyID">
  <xs:selector xpath="//*/>
  <xs:field xpath="@id"/>
</xs:unique>

```

XML representations for the three kinds of identity-constraint definitions.

#### 3.11.1. The Identity-constraint Definition Schema Component

The identity-constraint definition schema component has the following properties:

An NCName as defined by [\[XML-Namespaces\]](#). Either [absent](#) or a namespace name, as defined in [\[XML-Namespaces\]](#). One of key, keyref or unique. A restricted XPath ([\[XPath\]](#)) expression. A non-empty list of restricted XPath ([\[XPath\]](#)) expressions. Required if is keyref, forbidden otherwise. An identity-constraint definition with equal to key or unique. Optional. A set of annotations.

Identity-constraint definitions are identified by their and ; Identity-constraint definition identities must be unique within an [XML Schema](#). See § 4.2.3 – [References to schema components across namespaces](#) on page 110 for the use of component identifiers when importing one schema into another.

Informally, identifies the Identity-constraint definition as playing one of three roles:

- (unique) the Identity-constraint definition asserts uniqueness, with respect to the content identified by , of the tuples resulting from evaluation of the XPath expression(s).
- (key) the Identity-constraint definition asserts uniqueness as for unique. key further asserts that all selected content actually has such tuples.
- (keyref) the Identity-constraint definition asserts a correspondence, with respect to the content identified by , of the tuples resulting from evaluation of the XPath expression(s), with those of the .

These constraints are specified along side the specification of types for the attributes and elements involved, i.e. something declared as of type integer may also serve as a key. Each constraint declaration has a name, which exists in a single symbol space for constraints. The equality and inequality conditions appealed to in checking these constraints apply to the *value* of the fields selected, so that for example 3 . 0 and 3 would be conflicting keys if they were both number, but non-conflicting if they were both strings, or one was a string and one a number. Values of differing type can only be equal if one type is derived from the other, and the value is in the value space of both.


Overall the augmentations to XML's ID/IDREF mechanism are:

- Functioning as a part of an identity-constraint is in addition to, not instead of, having a type;
- Not just attribute values, but also element content and combinations of values and content can be declared to be unique;
- Identity-constraints are specified to hold within the scope of particular elements;
- (Combinations of) attribute values and/or element content can be declared to be keys, that is, not only unique, but always present and non-nullable;
- The comparison between keyref and key or unique is by value equality, not by string equality.

specifies a restricted XPath ([XPath]) expression relative to instances of the element being declared. This must identify a node set of subordinate elements (i.e. contained within the declared element) to which the constraint applies.

specifies XPath expressions relative to each element selected by a . This must identify a single node (element or attribute) whose content or value, which must be of a simple type, is used in the constraint. It is possible to specify an ordered list of s, to cater to multi-field keys, keyrefs, and uniqueness constraints.

In order to reduce the burden on implementers, in particular implementers of streaming processors, only restricted subsets of XPath expressions are allowed in and . The details are given in § 3.11.6 – [Constraints on Identity-constraint Definition Schema Components](#) on page 85.

 Provision for multi-field keys etc. goes beyond what is supported by `xs:1:key`.

See § 3.13 – [Annotations](#) on page 88 for information on the role of the property.

### 3.11.2. XML Representation of Identity-constraint Definition Schema Components

The XML representation for an identity-constraint definition schema component is either a , a or a element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

The **actual value** of the name attribute The **actual value** of the targetNamespace attribute of the parent schema element information item. One of key, keyref or unique, depending on the item. A restricted XPath expression corresponding to the **actual value** of the xpath attribute of the element information item among the children A sequence of XPath expressions, corresponding to the **actual values** of the xpath attributes of the element information item children, in order. If the item is a , the identity-constraint definition to by the **actual value** of the refer attribute, otherwise **absent**. The annotations corresponding to the element information item in the children, if present, and in the and children, if present, otherwise **absent**.



```

<xs:element name="vehicle">
  <xs:complexType>
    . . .
    <xs:attribute name="plateNumber" type="xs:integer"/>
    <xs:attribute name="state" type="twoLetterCode"/>
  </xs:complexType>
</xs:element>

<xs:element name="state">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="code" type="twoLetterCode"/>
      <xs:element ref="vehicle" maxOccurs="unbounded"/>
      <xs:element ref="person" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:key name="reg"> <!-- vehicles are keyed by their plate within states -->
    <xs:selector xpath="./vehicle"/>
    <xs:field xpath="@plateNumber"/>
  </xs:key>
</xs:element>

<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      . . .
      <xs:element ref="state" maxOccurs="unbounded"/>
      . . .
    </xs:sequence>
  </xs:complexType>

  <xs:key name="state"> <!-- states are keyed by their code -->
    <xs:selector xpath="./state"/>
    <xs:field xpath="code"/>
  </xs:key>

  <xs:keyref name="vehicleState" refer="state">
    <!-- every vehicle refers to its state -->
    <xs:selector xpath="./vehicle"/>
    <xs:field xpath="@state"/>
  </xs:keyref>

```



```

<xs:key name="regKey"> <!-- vehicles are keyed by a pair of state and plate -->
  <xs:selector xpath="//vehicle"/>
  <xs:field xpath="@state"/>
  <xs:field xpath="@plateNumber"/>
</xs:key>

<xs:keyref name="carRef" refer="regKey"> <!-- people's cars are a reference -->
  <xs:selector xpath="//car"/>
  <xs:field xpath="@regState"/>
  <xs:field xpath="@regPlate"/>
</xs:keyref>

</xs:element>

<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      . . .
      <xs:element name="car">
        <xs:complexType>
          <xs:attribute name="regState" type="twoLetterCode"/>
          <xs:attribute name="regPlate" type="xs:integer"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

A state element is defined, which contains a code child and some vehicle and person children. A vehicle in turn has a plateNumber attribute, which is an integer, and a state attribute. State's codes are a key for them within the document. Vehicle's plateNumbers are a key for them within states, and state and plateNumber is asserted to be a key for vehicle within the document as a whole. Furthermore, a person element has an empty car child, with regState and regPlate attributes, which are then asserted together to refer to vehicles via the carRef constraint. The requirement that a vehicle's state match its containing state's code is not expressed here.

### 3.11.3. Constraints on XML Representations of Identity-constraint Definitions

#### src: Identity-constraint Definition Representation OK

In addition to the conditions imposed on , and element information items by the schema for schemas, the corresponding identity-constraint definition must satisfy the conditions set out in § 3.11.6 – Constraints on Identity-constraint Definition Schema Components on page 85.

### 3.11.4. Identity-constraint Definition Validation Rules

#### cvc: Identity-constraint Satisfied

For an element information item to be locally **valid** with respect to an identity-constraint

1. The , with the element information item as the context node, evaluates to a node-set (as defined in [XPath]). Call this the *target node set*.

2. Each node in the [target node set](#) is either the context node or an element node among its descendants.
3. For each node in the [target node set](#) all of the  $\text{key-sequences}$ , with that node as the context node, evaluate to either an empty node-set or a node-set with exactly one member, which must have a simple type. Call the sequence of the type-determined values (as defined in [XML Schemas: Datatypes]) of the schema normalized value of the element and/or attribute information items in those node-sets in order the *key-sequence* of the node.
4. Call the subset of the [target node set](#) for which all the  $\text{key-sequences}$  evaluate to a node-set with exactly one member which is an element or attribute node with a simple type the *qualified node set*.
  - A. the is unique
 

no two members of the [qualified node set](#) have [key-sequences](#) whose members are pairwise equal, as defined by Equal in [XML Schemas: Datatypes].
  - B. the is key
    - i. The [target node set](#) and the [qualified node set](#) are equal, that is, every member of the [target node set](#) is also a member of the [qualified node set](#) and *vice versa*.
    - ii. No two members of the [qualified node set](#) have [key-sequences](#) whose members are pairwise equal, as defined by Equal in [XML Schemas: Datatypes].
    - iii. No element member of the [key-sequence](#) of any member of the [qualified node set](#) was assessed as [valid](#) by reference to an element declaration whose [is true](#).
  - C. the is keyref
 

for each member of the [qualified node set](#) (call this the keyref member), there must be a [node table](#) associated with the [keyref](#) in the [of the element information item](#) (see Identity-constraint Table An eligible identity-constraint of an element information item is one such that [or of is satisfied with respect to that item and that constraint](#), or such that any of the element information item children of that item have an [property](#) whose value has an entry for that constraint. A node table is a set of pairs each consisting of a [key-sequence](#) and an element node. Whenever an element information item has one or more eligible identity-constraints, in the post-schema-validation infoset that element information item has a [property](#) as follows: one Identity-constraint Binding information item for each eligible identity-constraint, with [properties](#) as follows: The eligible identity-constraint. A node table with one entry for every [key-sequence](#) (call it k) and node (call it n) such that [1. 2.](#) provided no two entries have the same [key-sequence](#) but distinct nodes. Potential conflicts are resolved by not including any conflicting entries which would have owed their inclusion to above. Note that if all the conflicting entries arose under above, this means no entry at all will appear for the offending [key-sequence](#). The complexity of the above arises from the fact that keyref identity-constraints may be defined on domains distinct from the embedded domain of the identity-constraint they reference, or the domains may be the same but self-embedding at some depth. In either case the node table for the referenced identity-constraint needs to propagate upwards, with conflict resolution. The Identity-constraint Binding information item, unlike others in this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of above. Accordingly, conformant processors may, but are not required to, expose them via [properties](#) in the post-schema-validation infoset. In other words, the above constraints may be read as saying validation of identity-constraints proceeds as if such infoset items existed. [, which must be understood as logically prior to this clause of this constraint](#), below) and there must be an entry

in that table whose [key-sequence](#) is equal to the keyref member's [key-sequence](#) member for member, as defined by Equal in [XML Schemas: Datatypes].



The use of schema normalized value in the definition of [key sequence](#) above means that default or fixed value constraints may play a part in [key sequences](#).



Because the validation of keyref (see ) depends on finding appropriate entries in a element information item's [node table](#), and [node tables](#) are assembled strictly recursively from the node tables of descendants, only element information items within the sub-tree rooted at the element information item being [validated](#) can be referenced successfully.



Although this specification defines a [post-schema-validation infoset](#) contribution which would enable schema-aware processors to implement above ( Element Declaration If an element information item is valid with respect to an element declaration as per then in the post-schema-validation infoset the element information item must, at processor option, have either: an item isomorphic to the declaration component itself or true if of above is satisfied, otherwise false ), processors are not required to provide it. This clause can be read as if in the absence of this infoset contribution, the value of the relevant property must be available.

### 3.11.5. Identity-constraint Definition Information Set Contributions

#### **sic: Identity-constraint Table**

An *eligible identity-constraint* of an element information item is one such that or of Identity-constraint Satisfied For an element information item to be locally valid with respect to an identity-constraint 1. 2. 3. 4. The use of schema normalized value in the definition of key sequence above means that default or fixed value constraints may play a part in key sequences. is satisfied with respect to that item and that constraint, or such that any of the element information item children of that item have an property whose value has an entry for that constraint.

A *node table* is a set of pairs each consisting of a [key-sequence](#) and an element node.

Whenever an element information item has one or more [eligible identity-constraints](#), in the [post-schema-validation infoset](#) that element information item has a property as follows:

one Identity-constraint Binding information item for each [eligible identity-constraint](#), with properties as follows: The [eligible identity-constraint](#). A [node table](#) with one entry for every [key-sequence](#) (call it k) and node (call it n) such that

1. There is an entry in one of the [node tables](#) associated with the in an Identity-constraint Binding information item in at least one of the s of the element information item children of the element information item whose [key-sequence](#) is k and whose node is n;
2. n appears with [key-sequence](#) k in the [qualified node set](#) for the .

provided no two entries have the same [key-sequence](#) but distinct nodes. Potential conflicts are resolved by not including any conflicting entries which would have owed their inclusion to above. Note that if all the conflicting entries arose under above, this means no entry at all will appear for the offending [key-sequence](#).



The complexity of the above arises from the fact that keyref identity-constraints may be defined on domains distinct from the embedded domain of the identity-constraint they reference, or the domains may be the same but self-

embedding at some depth. In either case the [node table](#) for the referenced identity-constraint needs to propagate upwards, with conflict resolution.

The Identity-constraint Binding information item, unlike others in this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of Identity-constraint Satisfied For an element information item to be locally valid with respect to an identity-constraint 1. 2. 3. 4. The use of schema normalized value in the definition of key sequence above means that default or fixed value constraints may play a part in key sequences. above. Accordingly, conformant processors may, but are *not* required to, expose them via properties in the [post-schema-validation infoset](#). In other words, the above constraints may be read as saying [validation](#) of identity-constraints proceeds *as if* such infoset items existed.

### 3.11.6. Constraints on Identity-constraint Definition Schema Components

All identity-constraint definitions (see § 3.11 – Identity-constraint Definitions on page 79) must satisfy the following constraint.

#### cos: Identity-constraint Definition Properties Correct

1. The values of the properties of an identity-constraint definition must be as described in the property tableau in § 3.11.1 – The Identity-constraint Definition Schema Component on page 79, modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. If the is keyref, the cardinality of the must equal that of the of the .

#### cos: Selector Value OK

1. The must be a valid XPath expression, as defined in [XPath].
2. A. It must conform to the following extended BNF:

##### Selector XPath expressions

] 1 [	Selector	=	:	:	Path ('' Path)*
] 2 [	Path	=	:	:	(./)? Step ('' Step)*
] 3 [	Step	=	:	:	'  NameTest
] 4 [	NameTest	=	:	:	QName   '*'   NCName ':' '*'

- B. It must be an XPath expression involving the `child` axis whose abbreviated form is as given above.

For readability, whitespace may be used in selector XPath expressions even though not explicitly allowed by the grammar: [whitespace](#) may be freely added within patterns before or after any [token](#).

#### Lexical productions

- [5] token ::= ' ' | '/' | '\\' | '|' | '@' | NameTest
- [6] whitespace ::= S

When tokenizing, the longest possible token is always returned.

**cos: Fields Value OK**

1. Each member of the must be a valid XPath expression, as defined in [XPath].
2. A. It must conform to the extended BNF given above for **Selector**, with the following modification:

**Path in Field XPath expressions**

] 7 [ Path = : : (./)? ( Step '/' )\* ( Step | '@' NameTest )

This production differs from the one above in allowing the final step to match an attribute node.

- B. It must be an XPath expression involving the `child` and/or `attribute` axes whose abbreviated form is as given above.

For readability, whitespace may be used in field XPath expressions even though not explicitly allowed by the grammar: **whitespace** may be freely added within patterns before or after any **token**.

When tokenizing, the longest possible token is always returned.

## 3.12. Notation Declarations

Notation declarations reconstruct XML 1.0 NOTATION declarations.



```
<xs:notation name="jpeg" public="image/jpeg" system="viewer.exe">
```

The XML representation of a notation declaration.

### 3.12.1. The Notation Declaration Schema Component

The notation declaration schema component has the following properties:

An NCName as defined by [XML-Namespaces]. Either **absent** or a namespace name, as defined in [XML-Namespaces]. Optional if is present. A URI reference. Optional if is present. A public identifier, as defined in [XML 1.0 (Second Edition)]. Optional. An annotation.

Notation declarations do not participate in **validation** as such. They are referenced in the course of **validating** strings as members of the NOTATION simple type.

See § 3.13 – Annotations on page 88 for information on the role of the property.

### 3.12.2. XML Representation of Notation Declaration Schema Components

The XML representation for a notation declaration schema component is a element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

The **actual value** of the `name` attribute The **actual value** of the `targetNamespace` attribute of the parent schema element information item. The **actual value** of the `system` attribute, if present, otherwise **absent**. The **actual value** of the `public` attribute The annotation corresponding to the element information item in the children, if present, otherwise **absent**.

```

☞ <xs:notation name="jpeg"
      public="image/jpeg" system="viewer.exe" />

<xs:element name="picture">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
        <xs:attribute name="pictype">
          <xs:simpleType>
            <xs:restriction base="xs:NOTATION">
              <xs:enumeration value="jpeg"/>
              <xs:enumeration value="png"/>
              . . .
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<picture pictype="jpeg">...</picture>

```

### 3.12.3. Constraints on XML Representations of Notation Declarations

#### src: Notation Definition Representation OK

In addition to the conditions imposed on element information items by the schema for schemas, the corresponding notation definition must satisfy the conditions set out in § 3.12.6 – [Constraints on Notation Declaration Schema Components](#) on page 88.

### 3.12.4. Notation Declaration Validation Rules

None as such.

### 3.12.5. Notation Declaration Information Set Contributions


#### sic: Validated with Notation

Whenever an attribute information item is [valid](#) with respect to a NOTATION, in the [post-schema-validation info](#)set its parent element information item either has a property as follows:

An [item isomorphic](#) to the notation declaration whose [local name](#) and [namespace name](#) (as defined in [QName Interpretation](#) Where the type of an attribute information item in a document involved in validation is identified as [QName](#), its actual value is composed of a local name and a namespace name. Its actual value is determined based on its normalized value and the containing element information item's in-scope namespaces following : [1. 2](#). In the absence of the in-scope namespaces property in the info set for the schema document in question, processors must reconstruct equivalent information as necessary, using the namespace attributes of the containing element information item and its ancestors. ) of the attribute item's [actual value](#)

or has a pair of properties as follows:

The value of the of that notation declaration. The value of the of that notation declaration.

 For compatibility, only one such attribute should appear on any given element. If more than one such attribute *does* appear, which one supplies the infoSet property or properties above is not defined.

### 3.12.6. Constraints on Notation Declaration Schema Components


All notation declarations (see § 3.12 – Notation Declarations on page 86) must satisfy the following constraint.

#### cos: Notation Declaration Correct

The values of the properties of a notation declaration must be as described in the property tableau in § 3.12.1 – The Notation Declaration Schema Component on page 86, modulo the impact of § 5.3 – Missing Sub-components on page 117.

## 3.13. Annotations

Annotations provide for human- and machine-targeted annotations of schema components.



```
<xs:simpleType fn:note="special">
  <xs:annotation>
    <xs:documentation>A type for experts only</xs:documentation>
    <xs:appinfo>
      <fn:specialHandling>checkForPrimes</fn:specialHandling>
    </xs:appinfo>
  </xs:annotation>
```

XML representations of three kinds of annotation.

### 3.13.1. The Annotation Schema Component

The annotation schema component has the following properties:

A sequence of element information items. A sequence of element information items. A sequence of attribute information items.

is intended for human consumption, for automatic processing. In both cases, provision is made for an optional URI reference to supplement the local information, as the value of the `source` attribute of the respective element information items. **Validation** does *not* involve dereferencing these URIs, when present. In the case of , indication should be given as to the identity of the (human) language used in the contents, using the `xml:lang` attribute.

ensures that when schema authors take advantage of the provision for adding attributes from namespaces other than the XML Schema namespace to schema documents, they are available within the components corresponding to the element items where such attributes appear.



Annotations do not participate in [validation](#) as such. Provided an annotation itself satisfies all relevant [Schema Component Constraints](#) it *cannot* affect the [validation](#) of element information items.

### 3.13.2. XML Representation of Annotation Schema Components

Annotation of schemas and schema components, with material for human or computer consumption, is provided for by allowing application information and human information at the beginning of most major schema elements, and anywhere at the top level of schemas. The XML representation for an annotation schema component is an element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

A sequence of the element information items from among the children, in order, if any, otherwise the empty sequence. A sequence of the element information items from among the children, in order, if any, otherwise the empty sequence. A sequence of attribute information items, namely those allowed by the attribute wildcard in the type definition for the item itself or for the enclosing items which correspond to the component within which the annotation component is located.

The annotation component corresponding to the element in the example above will have one element item in each of its and one attribute item in its .

### 3.13.3. Constraints on XML Representations of Annotations

#### **src: Annotation Definition Representation OK**

In addition to the conditions imposed on element information items by the schema for schemas, the corresponding annotation must satisfy the conditions set out in [§ 3.13.6 – Constraints on Annotation Schema Components](#) on page 89.

### 3.13.4. Annotation Validation Rules

None as such.

### 3.13.5. Annotation Information Set Contributions

None as such: the addition of annotations to the [post-schema-validation infoset](#) is covered by the [post-schema-validation infoset](#) contributions of the enclosing components.

### 3.13.6. Constraints on Annotation Schema Components

All annotations (see [§ 3.13 – Annotations](#) on page 88) must satisfy the following constraint.

#### **cos: Annotation Correct**


The values of the properties of an annotation must be as described in the property tableau in [§ 3.13.1 – The Annotation Schema Component](#) on page 88, modulo the impact of [§ 5.3 – Missing Sub-components](#) on page 117.

## 3.14. Simple Type Definitions



This section consists of a combination of non-normative versions of normative material from [\[XML Schemas: Datatypes\]](#), for local cross-reference purposes, and normative material relating to the interface between schema components defined in this specification and the simple type definition component.

Simple type definitions provide for constraining character information item children of element and attribute information items.



```
<xs:simpleType name="fahrenheitWaterTemp">
  <xs:restriction base="xs:number">
    <xs:fractionDigits value="2"/>
    <xs:minExclusive value="0.00"/>
    <xs:maxExclusive value="100.00"/>
  </xs:restriction>
</xs:simpleType>
```

The XML representation of a simple type definition.

### 3.14.1. (non-normative) The Simple Type Definition Schema Component

The simple type definition schema component has the following properties:

Optional. An NCName as defined by [XML-Namespaces]. Either [absent](#) or a namespace name, as defined in [XML-Namespaces]. A simple type definition, which may be the . A set of constraining facets. A set of fundamental facets. A subset of {extension, list, restriction, union}. One of {atomic, list, union}. Depending on the value of , further properties are defined as follows:

#### ***atomic***

A built-in primitive simple type definition.

#### ***list***


A simple type definition.

#### ***union***

A non-empty sequence of simple type definitions.

Optional. An annotation.

Simple types are identified by their and . Except for anonymous simple types (those with no ), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an [XML Schema](#), no simple type definition can have the same name as another simple or complex type definition. Simple type *s* and *s* are provided for reference from instances (see § 2.6.1 – [xsi:type](#) on page 12), and for use in the XML representation of schema components (specifically in and ). See § 4.2.3 – [References to schema components across namespaces](#) on page 110 for the use of component identifiers when importing one schema into another.

 The of a simple type is not *ipso facto* the (local) name of the element or attribute information items [validated](#) by that definition. The connection between a name and a type definition is described in § 3.3 – [Element Declarations](#) on page 22 and § 3.2 – [Attribute Declarations](#) on page 16.

A simple type definition with an empty specification for can be used as the for other types derived by either of extension or restriction, or as the in the definition of a list, or in the of a union; the explicit values extension, restriction, list and union prevent further derivations by extension (to yield a complex type) and restriction (to yield a simple type) and use in constructing lists and unions respectively.

determines whether the simple type corresponds to an atomic, list or union type as defined by [XML Schemas: Datatypes].

As described in § 2.2.1.1 – [Type Definition Hierarchy](#) on page 5, every simple type definition is a [restriction](#) of some other simple type (the `base`), which is the [simple ur-type definition](#) if and only if the type definition in question is one of the built-in primitive datatypes, or a list or union type definition which is not itself derived by restriction from a list or union respectively. Each *atomic* type is ultimately a restriction of exactly one such built-in primitive datatype, which is its `base`.

For each simple type definition, the `value` and `lexical` spaces are selected from those defined in [[XML Schemas: Datatypes](#)]. For atomic definitions, these are restricted to those appropriate for the corresponding `base`. Therefore, the value space and lexical space (i.e. what is [validated](#) by any atomic simple type) is determined by the pair (`base`, `value`).

As specified in [[XML Schemas: Datatypes](#)], list simple type definitions [validate](#) space separated tokens, each of which conforms to a specified simple type definition, the `value`. The item type specified must not itself be a list type, and must be one of the types identified in [[XML Schemas: Datatypes](#)] as a suitable item type for a list simple type. In this case the `value` applies to the list itself, and are restricted to those appropriate for lists.

A union simple type definition [validates](#) strings which satisfy at least one of its `value`. As in the case of list, the `value` applies to the union itself, and are restricted to those appropriate for unions.

The [simple ur-type definition](#) must *not* be named as the `base` of any user-defined atomic simple type definitions: as it has no constraining facets, this would be incoherent.

See § 3.13 – [Annotations](#) on page 88 for information on the role of the `property`.

### 3.14.2. (non-normative) XML Representation of Simple Type Definition Schema Components

 This section reproduces a version of material from [[XML Schemas: Datatypes](#)], for local cross-reference purposes.

The [actual value](#) of the `name` attribute if present, otherwise [absent](#). The [actual value](#) of the `targetNamespace` attribute of the ancestor element information item if present, otherwise [absent](#).

1. the `alternative` is chosen

the type definition to be by the [actual value](#) of the `base` attribute of `base`, if present, otherwise the type definition corresponding to the `base` among the children of `base`.

2. the `or` alternative is chosen

the `value`.

As for the `property` of complex type definitions, but using the `final` and `finalDefault` attributes in place of the `block` and `blockDefault` attributes and with the relevant set being {`extension`, `restriction`, `list`, `union`}. If the `alternative` is chosen, then `list`, otherwise if the `alternative` is chosen, then `union`, otherwise (the `alternative` is chosen), then the `value` of the `base`.

If the `base` is atomic, the following additional property mappings also apply:

The built-in primitive type definition from which the `base` is derived. A set of facet components [constituting a restriction](#) of the `value` of the `base` with respect to a set of facet components corresponding to the appropriate element information items among the children of `base` (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S) 1. 2. 3. If above holds, the `value` of R constitute a restriction of the `value` of B with respect to S. .

If the `base` is list, the following additional property mappings also apply:

1. the `alternative` is chosen

the type definition to by the [actual value](#) of the `itemType` attribute of , if present, otherwise the type definition corresponding to the among the children of .

2. the option is chosen

the of the .

If the alternative is chosen, a set of facet components [constituting a restriction](#) of the of the with respect to a set of facet components corresponding to the appropriate element information items among the children of (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S) 1. 2. 3. If above holds, the of R constitute a restriction of the of B with respect to S. , otherwise the empty set.

If the is union, the following additional property mappings also apply:

1. the alternative is chosen

define the *explicit members* as the type definitions to by the items in the [actual value](#) of the `memberTypes` attribute, if any, followed by the type definitions corresponding to the s among the children of , if any. The actual value is then formed by replacing any union type definition in the [explicit members](#) with the members of their , in order.

2. the option is chosen

the of the .

If the alternative is chosen, a set of facet components [constituting a restriction](#) of the of the with respect to a set of facet components corresponding to the appropriate element information items among the children of (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S) 1. 2. 3. If above holds, the of R constitute a restriction of the of B with respect to S. , otherwise the empty set.

### 3.14.3. Constraints on XML Representations of Simple Type Definitions

#### src: Simple Type Definition Representation OK

In addition to the conditions imposed on element information items by the schema for schemas,

1. The corresponding simple type definition, if any, must satisfy the conditions set out in § 3.14.6 – [Constraints on Simple Type Definition Schema Components](#) on page 93.
2. If the alternative is chosen, either it must have a `base` attribute or a among its children, but not both.
3. If the alternative is chosen, either it must have an `itemType` attribute or a among its children, but not both.
4. Circular union type definition is disallowed. That is, if the alternative is chosen, there must not be any entries in the `memberTypes` attribute at any depth which resolve to the component corresponding to the .

### 3.14.4. Simple Type Definition Validation Rules

#### **cvc: String Valid**

For a string to be locally **valid** with respect to a simple type definition

1. It is schema-valid with respect to that definition as defined by Datatype Valid in [XML Schemas: Datatypes].
2. A. The definition is ENTITY or is validly derived from ENTITY given the empty set, as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2.  
the string must be a **declared entity name**.
- B. The definition is ENTITIES or is validly derived from ENTITIES given the empty set, as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2.  
every whitespace-delimited substring of the string must be a **declared entity name**.
- C. no further condition applies.

A string is a *declared entity name* if it is equal to the name of some unparsed entity information item in the value of the unparsedEntities property of the document information item at the root of the infoset containing the element or attribute information item whose **normalized value** the string is.

### 3.14.5. Simple Type Definition Information Set Contributions

None as such.

### 3.14.6. Constraints on Simple Type Definition Schema Components

All simple type definitions other than the and the built-in primitive datatype definitions (see § 3.14 – Simple Type Definitions on page 89) must satisfy both the following constraints.

#### **cos: Simple Type Definition Properties Correct**

1. The values of the properties of a simple type definition must be as described in the property tableau in [Datatype definition](#), modulo the impact of § 5.3 – Missing Sub-components on page 117.
2. All simple type definitions must be derived ultimately from the circular definitions are disallowed). That is, it must be possible to reach a built-in primitive datatype or the by repeatedly following the .
3. The of the must not contain restriction.

#### **cos: Derivation Valid (Restriction, Simple)**

1. the is atomic
  - A. The must be an atomic simple type definition or a built-in primitive datatype.
  - B. The of the must not contain restriction.

- C. For each facet in the (call this DF)
    - i. DF must be an allowed constraining facet for the , as specified in the appropriate subsection of [3.2 Primitive datatypes](#).
    - ii. If there is a facet of the same kind in the of the (call this BF), then the DF's value must be a valid restriction of BF's value as defined in [[XML Schemas: Datatypes](#)].
2. the is list
- A. The must have a of atomic or union (in which case all the must be atomic).
  - B.
  - C. i. the is the
    - a) The of the must not contain list.
    - b) The must only contain the whiteSpace facet component.
  - ii. a) The must have a of list.
    - b) The of the must not contain restriction.
    - c) The must be validly derived from the 's given the empty set, as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) [1. 2. .](#)
    - d) Only length, minLength, maxLength, whiteSpace, pattern and enumeration facet components are allowed among the .
    - e) For each facet in the (call this DF), if there is a facet of the same kind in the of the (call this BF), then the DF's value must be a valid restriction of BF's value as defined in [[XML Schemas: Datatypes](#)].

The first case above will apply when a list is derived by specifying an item type, the second when derived by restriction from another list.

3. the is union
- A. The must all have of atomic or list.
  - B.
  - C. i. the is the
    - a) All of the must have a which does not contain union.
    - b) The must be empty.
  - ii. a) The must have a of union.
    - b) The of the must not contain restriction.
    - c) The , in order, must be validly derived from the corresponding type definitions in the 's given the empty set, as defined in Type Derivation OK (Simple) For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base)

given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant) 1. 2. .

- d) Only pattern and enumeration facet components are allowed among the .
- e) For each facet in the (call this DF), if there is a facet of the same kind in the of the (call this BF), then the DF's value must be a valid restriction of BF's value as defined in [XML Schemas: Datatypes].

The first case above will apply when a union is derived by specifying one or more member types, the second when derived by restriction from another union.

If this constraint Derivation Valid (Restriction, Simple) 1. 2. 3. If this constraint holds of a simple type definition, it is a valid restriction of its base type definition. holds of a simple type definition, it is a *valid restriction* of its .

The following constraint defines relations appealed to elsewhere in this specification.

#### cos: Type Derivation OK (Simple)

For a simple type definition (call it D, for derived) to be validly derived from a type definition (call this B, for base) given a subset of {extension, restriction, list, union} (of which only restriction is actually relevant)

1. They are the same type definition.
2. A. restriction is not in the subset, or in the of its own ;
  - B. i. D's is B.
  - ii. D's is not the and is validly derived from B given the subset, as defined by this constraint.
  - iii. D's is list or union and B is the .
  - iv. B's is union and D is validly derived from a type definition in B's given the subset, as defined by this constraint.



With respect to , see the Note on identity at the end of The wording of above appeals to a notion of component identity which is only incompletely defined by this version of this specification. In some cases, the wording of this specification does make clear the rules for component identity. These cases include: When they are both top-level components with the same component type, namespace name, and local name; When they are necessarily the same type definition (for example, when the two types definitions in question are the type definitions associated with two attribute or element declarations, which are discovered to be the same declaration); When they are the same by construction (for example, when an element's type definition defaults to being the same type definition as that of its substitution-group head or when a complex type definition inherits an attribute declaration from its base type definition). In other cases two conforming implementations may disagree as to whether components are identical. above.

#### cos: Simple Type Restriction (Facets)

For a simple type definition (call it R) to restrict another simple type definition (call it B) with a set of facets (call this S)

1. The of R is the same as that of B.



2. If is atomic, the of R is the same as that of B.
3. The of R are the union of S and the of B, eliminating duplicates. To eliminate duplicates, when a facet of the same kind occurs in both S and the of B, the one in the of B is not included, with the exception of enumeration and pattern facets, for which multiple occurrences with distinct values are allowed.

Additional constraint(s) may apply depending on the kind of facet, see the appropriate sub-section of [4.3 Constraining Facets](#)

If above holds, the of R *constitute a restriction* of the of B with respect to S.

### 3.14.7. Built-in Simple Type Definition

There is a simple type definition nearly equivalent to the [simple ur-type definition](#) present in every schema by definition. It has the following properties:

Simple Type Definition of the Ur-Type anySimpleType <http://www.w3.org/2001/XMLSchema> The empty set [absent](#)


The [simple ur-type definition](#) is the root of the simple type definition hierarchy, and as such mediates between the other simple type definitions, which all eventually trace back to it via their properties, and the [ur-type definition](#), which is *its* . This is why the is exempted from the first clause of Simple Type Definition Properties Correct 1. 2. 3. , which would otherwise bar it because of its derivation from a complex type definition and absence of .

Simple type definitions for all the built-in primitive datatypes, namely string, boolean, float, double, number, dateTime, duration, time, date, gMonth, gMonthDay, gDay, gYear, gYearMonth, hexBinary, base64Binary, anyURI (see the Primitive Datatypes section of [\[XML Schemas: Datatypes\]](#)) are present by definition in every schema. All are in the XML Schema (namespace name <http://www.w3.org/2001/XMLSchema>), have an atomic with an empty and the [simple ur-type definition](#) as their and themselves as .

Similarly, simple type definitions for all the built-in derived datatypes (see the Derived Datatypes section of [\[XML Schemas: Datatypes\]](#)) are present by definition in every schema, with properties as specified in [\[XML Schemas: Datatypes\]](#) and as represented in XML in [Appendix A – Schema for Schemas \(normative\)](#) on page 118.

## 3.15. Schemas as a Whole

A schema consists of a set of schema components.



```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/example">
  . . .
</xs:schema>
```

The XML representation of the skeleton of a schema.

### 3.15.1. The Schema Itself

At the abstract level, the schema itself is just a container for its components.

A set of named simple and complex type definitions. A set of named (top-level) attribute declarations. A set of named (top-level) element declarations. A set of named attribute group definitions. A set of named model group definitions. A set of notation declarations. A set of annotations.


### 3.15.2. XML Representations of Schemas

A schema is represented in XML by one or more [schema documents](#), that is, one or more element information items. A [schema document](#) contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. A [schema document](#) which has one or more element information items corresponds to a schema with components with more than one target namespace, see [Import Constraints and Semantics](#) In addition to the conditions imposed on element information items by the schema for schemas 1. 2. 3. It is not an error for the application schema reference strategy to fail. It is an error for it to resolve but the rest of above to fail to be satisfied. Failure to find a referent may well cause less than complete assessment outcomes, of course. The schema components (that is , , , , ) of a schema corresponding to a element information item with one or more element information items must include not only definitions or declarations corresponding to the appropriate members of its children, but also, for each of those element information items for which above is satisfied, a set of schema components identical to all the schema components of I. .

The simple and complex type definitions corresponding to all the and element information items in the children, if any, plus any included or imported definitions, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The (top-level) attribute declarations corresponding to all the element information items in the children, if any, plus any included or imported declarations, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The (top-level) element declarations corresponding to all the element information items in the children, if any, plus any included or imported declarations, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The attribute group definitions corresponding to all the element information items in the children, if any, plus any included or imported definitions, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The model group definitions corresponding to all the element information items in the children, if any, plus any included or imported definitions, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The notation declarations corresponding to all the element information items in the children, if any, plus any included or imported declarations, see [§ 4.2.1 – Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105 and [§ 4.2.3 – References to schema components across namespaces](#) on page 110. The annotations corresponding to all the element information items in the children, if any. Note that none of the attribute information items displayed above correspond directly to properties of schemas. The `blockDefault`, `finalDefault`, `attributeFormDefault`, `elementFormDefault` and `targetNamespace` attributes are appealed to in the sub-sections above, as they provide global information applicable to many representation/component correspondences. The other attributes (`id` and `version`) are for user convenience, and this specification defines no semantics for them.

The definition of the schema abstract data model in [§ 2.2 – XML Schema Abstract Data Model](#) on page 4 makes clear that most components have a target namespace. Most components corresponding to representations within a given element information item will have a target namespace which corresponds to the `targetNamespace` attribute.

Since the empty string is not a legal namespace name, supplying an empty string for `targetNamespace` is incoherent, and is *not* the same as not specifying it at all. The appropriate form of schema document corresponding to a [schema](#) whose components have no is one which has no `targetNamespace` attribute specified at all.

 The XML namespaces Recommendation discusses only instance document syntax for elements and attributes; it therefore provides no direct framework for managing the names of type definitions, attribute group definitions, and so on. Nevertheless, the specification applies the target namespace facility uniformly to all schema components, i.e. not only declarations but also definitions have a target namespace.

Although the example schema at the beginning of this section might be a complete XML document, need not be the document element, but can appear within other documents. Indeed there is no requirement that a schema correspond to a (text) document at all: it could correspond to an element information item constructed 'by hand', for instance via a DOM-conformant API.

Aside from and , which do not correspond directly to any schema component at all, each of the element information items which may appear in the content of corresponds to a schema component, and all except are named. The sections below present each such item in turn, setting out the components to which it may correspond.

### 3.15.2.1. References to Schema Components

Reference to schema components from a schema document is managed in a uniform way, whether the component corresponds to an element information item from the same schema document or is imported (§ 4.2.3 – [References to schema components across namespaces](#) on page 110) from an external schema (which may, but need not, correspond to an actual schema document). The form of all such references is a [QName](#).

A *QName* is a name with an optional namespace qualification, as defined in [[XML-Namespaces](#)]. When used in connection with the XML representation of schema components or references to them, this refers to the simple type QName as defined in [[XML Schemas: Datatypes](#)].

An *NCName* is a name with no colon, as defined in [[XML-Namespaces](#)]. When used in connection with the XML representation of schema components in this specification, this refers to the simple type NCName as defined in [[XML Schemas: Datatypes](#)].

In each of the XML representation expositions in the following sections, an attribute is shown as having type QName if and only if it is interpreted as referencing a schema component.



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:xhtml="http://www.w3.org/1999/xhtml"
          xmlns="http://www.example.com"
          targetNamespace="http://www.example.com">
  . . .
  <xs:element name="elem1" type="Address"/>
  <xs:element name="elem2" type="xhtml:blockquote"/>
  <xs:attribute name="attr1"
                type="xsl:quantity"/>
  . . .
```

```
</xs:schema>
```

The first of these is most probably a local reference, i.e. a reference to a type definition corresponding to a element information item located elsewhere in the schema document, the other two refer to type definitions from schemas for other namespaces and assume that their namespaces have been declared for import. See § 4.2.3 – [References to schema components across namespaces](#) on page 110 for a discussion of importing.

### 3.15.2.2. References to Schema Components from Elsewhere

The names of schema components such as type definitions and element declarations are not of type ID: they are not unique within a schema, just within a symbol space. This means that simple fragment identifiers will not always work to reference schema components from outside the context of schema documents.

There is currently no provision in the definition of the interpretation of fragment identifiers for the `text/xml` MIME type, which is the MIME type for schemas, for referencing schema components as such. However, [XPointer] provides a mechanism which maps well onto the notion of symbol spaces as it is reflected in the XML representation of schema components. A fragment identifier of the form `#xpointer(xs:schema/xs:element[@name="person"])` will uniquely identify the representation of a top-level element declaration with name `person`, and similar fragment identifiers can obviously be constructed for the other global symbol spaces.

Short-form fragment identifiers may also be used in some cases, that is when a DTD or XML Schema is available for the schema in question, and the provision of an `id` attribute for the representations of all primary and secondary schema components, which *is* of type ID, has been exploited.

It is a matter for applications to specify whether they interpret document-level references of either of the above varieties as being to the relevant element information item (i.e. without special recognition of the relation of schema documents to schema components) or as being to the corresponding schema component.

### 3.15.3. Constraints on XML Representations of Schemas

#### src: QName Interpretation

Where the type of an attribute information item in a document involved in [validation](#) is identified as [QName](#), its [actual value](#) is composed of a *local name* and a *namespace name*. Its [actual value](#) is determined based on its [normalized value](#) and the containing element information item's in-scope namespaces following [XML-Namespaces]:

1. its [normalized value](#) is prefixed
  - A. There must be a namespace in the in-scope namespaces whose prefix matches the prefix.
  - B. its [namespace name](#) is the namespace name of that namespace.
  - C. Its [local name](#) is the portion of its [normalized value](#) after the colon (':').
2. (its [normalized value](#) is unprefixd)
  - A. its [local name](#) is its [normalized value](#).
  - B.
    - i. there is a namespace in the in-scope namespaces whose prefix has no value  
its [namespace name](#) is the namespace name of that namespace.
    - ii. its [namespace name](#) is [absent](#).

In the absence of the in-scope namespaces property in the infoset for the schema document in question, processors must reconstruct equivalent information as necessary, using the namespace attributes of the containing element information item and its ancestors.

Whenever the word *resolve* in any form is used in this chapter in connection with a [QName](#) in a schema document, the following definition QName resolution (Schema Document) For a QName to resolve to a schema component of a specified kind 1. 2. 3. 4. should be understood:

**src: QName resolution (Schema Document)**

For a [QName](#) to resolve to a schema component of a specified kind

1. That component is a member of the value of the appropriate property of the schema which corresponds to the schema document within which the [QName](#) appears, that is
  - A. the kind specified is simple or complex type definition  
the property is the .
  - B. the kind specified is attribute declaration  
the property is the .
  - C. the kind specified is element declaration  
the property is the .
  - D. the kind specified is attribute group  
the property is the .
  - E. the kind specified is model group  
the property is the .
  - F. the kind specified is notation declaration  
the property is the .
2. The component's name matches the [local name](#) of the [QName](#);
3. The component's target namespace is identical to the [namespace name](#) of the [QName](#);
4.
  - A. the [namespace name](#) of the [QName](#) is [absent](#)
    - i. The element information item of the schema document containing the [QName](#) has no `targetNamespace` attribute.
    - ii. The element information item of the that schema document contains an element information item with no `namespace` attribute.
  - B. the [namespace name](#) of the [QName](#) is the same as
    - i. The [actual value](#) of the `targetNamespace` attribute of the element information item of the schema document containing the [QName](#).
    - ii. The [actual value](#) of the `namespace` attribute of some element information item contained in the element information item of that schema document.

### 3.15.4. Validation Rules for Schemas as a Whole

As the discussion above at § 3 – [Schema Component Details](#) on page 13 makes clear, at the level of schema components and [validation](#), reference to components by name is normally not involved. In a few cases, however, qualified names appearing in information items being [validated](#) must be resolved to schema components by such lookup. The following constraint is appealed to in these cases.

#### **cvc: QName resolution (Instance)**

A pair of a local name and a namespace name (or [absent](#)) resolve to a schema component of a specified kind in the context of [validation](#) by appeal to the appropriate property of the schema being used for the [assessment](#). Each such property indexes components by name. The property to use is determined by the kind of component specified, that is,

1. the kind specified is simple or complex type definition  
the property is the .
2. the kind specified is attribute declaration  
the property is the .
3. the kind specified is element declaration  
the property is the .
4. the kind specified is attribute group  
the property is the .
5. the kind specified is model group  
the property is the .
6. the kind specified is notation declaration  
the property is the .

The component resolved to is the entry in the table whose name matches the local name of the pair and whose target namespace is identical to the namespace name of the pair.

### 3.15.5. Schema Information Set Contributions

#### **sic: Schema Information**

Schema components provide a wealth of information about the basis of [assessment](#), which may well be of relevance to subsequent processing. Reflecting component structure into a form suitable for inclusion in the [post-schema-validation infoset](#) is the way this specification provides for making this information available.

Accordingly, by an *item isomorphic* to a component is meant an information item whose type is equivalent to the component's, with one property per property of the component, with the same name, and value either the same atomic value, or an information item corresponding in the same way to its component value, recursively, as necessary.

Processors must add a property in the [post-schema-validation infoset](#) to the element information item at which [assessment](#) began, as follows:

A set of namespace schema information information items, one for each namespace name which appears as the target namespace of any schema component in the schema used for that assessment, and one for [absent](#) if any schema component in the schema had no target namespace. Each namespace schema information information item has the following properties and values: A namespace name or [absent](#). A (possibly empty) set of schema component information items, each one an [item isomorphic](#) to a component whose target namespace is the sibling property above, drawn from the schema used for [assessment](#). A (possibly empty) set of schema document information items, with properties and values as follows, for each schema document which contributed components to the schema, and whose `targetNamespace` matches the sibling property above (or whose `targetNamespace` was [absent](#) but that contributed components to that namespace by being d by a schema document with that `targetNamespace` as per § 4.2.1 – [Assembling a schema for a single target namespace from multiple schema definition documents](#) on page 105): Either a URI reference, if available, otherwise [absent](#) A document information item, if available, otherwise [absent](#).

The property is provided for processors which wish to provide a single access point to the components of the schema which was used during [assessment](#). Lightweight processors are free to leave it empty, but if it *is* provided, it must contain at a minimum all the top-level (i.e. named) components which actually figured in the [assessment](#), either directly or (because an anonymous component which figured is contained within) indirectly.

#### **sic: ID/IDREF Table**

In the [post-schema-validation info set](#) a set of ID/IDREF binding information items is associated with the [validation root](#) element information item:

A (possibly empty) set of ID/IDREF binding information items, as specified below.

Let the *eligible item set* be the set of consisting of every attribute or element information item for which

1. its validation context is the [validation root](#);
2. it was successfully [validated](#) with respect to an attribute declaration as per Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration 1. 2. 3. 4. or element declaration as per Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. (as appropriate) whose attribute or element (respectively) is the built-in ID, IDREF or IDREFS simple type definition or a type derived from one of them.

Then there is one ID/IDREF binding in the for every distinct string which is

1. the [actual value](#) of a member of the [eligible item set](#) whose type definition is or is derived from ID or IDREF;
2. one of the items in the [actual value](#) of a member of the [eligible item set](#) whose type definition is or is derived from IDREFS.

Each ID/IDREF binding has properties as follows:

The string identified above. A set consisting of every element information item for which

1. its is the [validation root](#);
2. it has an attribute information item in its attributes or an element information item in its children which was [validated](#) by the built-in ID simple type definition or a type derived from it whose schema normalized value is the of this ID/IDREF binding.



The net effect of the above is to have one entry for every string used as an id, whether by declaration or by reference, associated with those elements, if any, which actually purport to have that id. See Validation Root Valid (ID/IDREF) For an element information item which is the validation root to be valid 1. 2. See for the definition of ID/IDREF binding. The first clause above applies when there is a reference to an undefined ID. The second applies when there is a multiply-defined ID. They are separated out to ensure that distinct error codes (see ) are associated with these two cases. Although this rule applies at the validation root, in practice processors, particularly streaming processors, may wish to detect and signal the case as it arises. This reconstruction of 's ID/IDREF functionality is imperfect in that if the validation root is not the document element of an XML document, the results will not necessarily be the same as those a validating parser would give were the document to have a DTD with equivalent declarations. above for the validation rule which actually checks for errors here.



The ID/IDREF binding information item, unlike most other aspects of this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of Validation Root Valid (ID/IDREF) For an element information item which is the validation root to be valid 1. 2. See for the definition of ID/IDREF binding. The first clause above applies when there is a reference to an undefined ID. The second applies when there is a multiply-defined ID. They are separated out to ensure that distinct error codes (see ) are associated with these two cases. Although this rule applies at the validation root, in practice processors, particularly streaming processors, may wish to detect and signal the case as it arises. This reconstruction of 's ID/IDREF functionality is imperfect in that if the validation root is not the document element of an XML document, the results will not necessarily be the same as those a validating parser would give were the document to have a DTD with equivalent declarations. above. Accordingly, conformant processors may, but are *not* required to, expose it in the [post-schema-validation infoset](#). In other words, the above constraint may be read as saying [assessment](#) proceeds *as if* such an infoset item existed.

### 3.15.6. Constraints on Schemas as a Whole

All schemas (see § 3.15 – Schemas as a Whole on page 96) must satisfy the following constraint.

#### cos: Schema Properties Correct

1. The values of the properties of a schema must be as described in the property tableau in § 3.15.1 – [The Schema Itself](#) on page 96, modulo the impact of § 5.3 – [Missing Sub-components](#) on page 117;
2. Each of the , , , and must not contain two or more schema components with the same name and target namespace.

## 4. Schemas and Namespaces: Access and Composition

This chapter defines the mechanisms by which this specification establishes the necessary precondition for [assessment](#), namely access to one or more schemas. This chapter also sets out in detail the relationship between schemas and namespaces, as well as mechanisms for modularization of schemas, including provision for incorporating definitions and declarations from one schema in another, possibly with modifications.

§ 2.4 – [Conformance](#) on page 10 describes three levels of conformance for schema processors, and § 5 – [Schemas and Schema-validity Assessment](#) on page 115 provides a formal definition of [assessment](#). This section sets out in detail the 3-layer architecture implied by the three conformance levels. The layers are:

1. The [assessment](#) core, relating schema components and instance information items;

2. Schema representation: the connections between XML representations and schema components, including the relationships between namespaces and schema components;
3. XML Schema web-interoperability guidelines: instance->schema and schema->schema connections for the WWW.

Layer 1 specifies the manner in which a schema composed of schema components can be applied to in the [assessment](#) of an instance element information item. Layer 2 specifies the use of elements in XML documents as the standard XML representation for schema information in a broad range of computer systems and execution environments. To support interoperation over the World Wide Web in particular, layer 3 provides a set of conventions for schema reference on the Web. Additional details on each of the three layers is provided in the sections below.

## 4.1. Layer 1: Summary of the Schema-validity Assessment Core

The fundamental purpose of the [assessment](#) core is to define [assessment](#) for a single element information item and its descendants with respect to a complex type definition. All processors are required to implement this core predicate in a manner which conforms exactly to this specification.

[assessment](#) is defined with reference to an [XML Schema](#) (note *not* a [schema document](#)) which consists of (at a minimum) the set of schema components (definitions and declarations) required for that [assessment](#). This is not a circular definition, but rather a *post facto* observation: no element information item can be fully assessed unless all the components required by any aspect of its (potentially recursive) [assessment](#) are present in the schema.

As specified above, each schema component is associated directly or indirectly with a target namespace, or explicitly with no namespace. In the case of multi-namespace documents, components for more than one target namespace will co-exist in a schema.

Processors have the option to assemble (and perhaps to optimize or pre-compile) the entire schema prior to the start of an [assessment](#) episode, or to gather the schema lazily as individual components are required. In all cases it is required that:

- The processor succeed in locating the [schema components](#) transitively required to complete an [assessment](#) (note that components derived from [schema documents](#) can be integrated with components obtained through other means);
- no definition or declaration changes once it has been established;
- if the processor chooses to acquire declarations and definitions dynamically, that there be no side effects of such dynamic acquisition that would cause the results of [assessment](#) to differ from that which would have been obtained from the same schema components acquired in bulk.




the [assessment](#) core is defined in terms of schema components at the abstract level, and no mention is made of the schema definition syntax (i.e. ). Although many processors will acquire schemas in this format, others may operate on compiled representations, on a programmatic representation as exposed in some programming language, etc.


The obligation of a schema-aware processor as far as the [assessment](#) core is concerned is to implement one or more of the options for [assessment](#) given below in § 5.2 – [Assessing Schema-Validity](#) on page 116. Neither the choice of element information item for that [assessment](#), nor which of the means of initiating [assessment](#) are used, is within the scope of this specification.

Although [assessment](#) is defined recursively, it is also intended to be implementable in streaming processors. Such processors may choose to incrementally assemble the schema during processing in response, for example, to encountering new namespaces. The implication of the invariants expressed above is that such incremental assembly must result in an [assessment](#) outcome that is the *same* as would be given if [assessment](#) was undertaken again with the final, fully assembled schema.

## 4.2. Layer 2: Schema Documents, Namespaces and Composition

The sub-sections of § 3 – [Schema Component Details](#) on page 13 define an XML representation for type definitions and element declarations and so on, specifying their target namespace and collecting them into schema documents. The two following sections relate to assembling a complete schema for [assessment](#) from multiple sources. They should *not* be understood as a form of text substitution, but rather as providing mechanisms for distributed definition of schema components, with appropriate schema-specific semantics.

 The core [assessment](#) architecture requires that a complete schema with all the necessary declarations and definitions be available. This may involve resolving both instance->schema and schema->schema references. As observed earlier in § 2.4 – [Conformance](#) on page 10, the precise mechanisms for resolving such references are expected to evolve over time. In support of such evolution, this specification observes the design principle that references from one schema document to a schema use mechanisms that directly parallel those used to reference a schema from an instance document.

 In the sections below, "schemaLocation" really belongs at layer 3. For convenience, it is documented with the layer 2 mechanisms of import and include, with which it is closely associated.

### 4.2.1. Assembling a schema for a single target namespace from multiple schema definition documents

Schema components for a single target namespace can be assembled from several [schema documents](#), that is several element information items:

A information item may contain any number of elements. Their `schemaLocation` attributes, consisting of a URI reference, identify other [schema documents](#), that is information items.

The [XML Schema](#) corresponding to contains not only the components corresponding to its definition and declaration children, but also all the components of all the [XML Schemas](#) corresponding to any d schema documents. Such included schema documents must either (a) have the same `targetNamespace` as the ing schema document, or (b) no `targetNamespace` at all, in which case the d schema document is converted to the ing schema document's `targetNamespace`.

#### **src: Inclusion Constraints and Semantics**

In addition to the conditions imposed on element information items by the schema for schemas,

1. If the [actual value](#) of the `schemaLocation` attribute successfully resolves
  - A. It resolves to (a fragment of) a resource which is an XML document (of type `application/xml` or `text/xml` with an XML declaration for preference, but this is not required), which in turn corresponds to a element information item in a well-formed information set, which in turn corresponds to a valid schema.
  - B. It resolves to a element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the *d* item SII, the valid schema I and the *ing* item's parent item SII'.

2. A. SII has a `targetNamespace` attribute, and its **actual value** is identical to the **actual value** of the `targetNamespace` attribute of SII' (which must have such an attribute).
  - B. Neither SII nor SII' have a `targetNamespace` attribute.
  - C. SII has no `targetNamespace` attribute (but SII' does).
3. A. or above is satisfied

the schema corresponding to SII' must include not only definitions or declarations corresponding to the appropriate members of its own children, but also components identical to all the **schema components** of I.

- B. above is satisfied

the schema corresponding to the *d* item's parent must include not only definitions or declarations corresponding to the appropriate members of its own children, but also components identical to all the **schema components** of I, except that anywhere the **absent** target namespace name would have appeared, the **actual value** of the `targetNamespace` attribute of SII' is used. In particular, it replaces **absent** in the following places:

- i. The target namespace of named schema components, both at the top level and (in the case of nested type definitions and nested attribute and element declarations whose `code` was qualified) nested within definitions;
- ii. The of a wildcard, whether negated or not;

It is *not* an error for the **actual value** of the `schemaLocation` attribute to fail to resolve it all, in which case no corresponding inclusion is performed. It *is* an error for it to resolve but the rest of clause 1 above to fail to be satisfied. Failure to resolve may well cause less than complete **assessment** outcomes, of course.

As discussed in § 5.3 – **Missing Sub-components** on page 117, **QNames** in XML representations may fail to **resolve**, rendering components incomplete and unusable because of missing subcomponents. During schema construction, implementations must retain **QName** values for such references, in case an appropriately-named component becomes available to discharge the reference by the time it is actually needed. **Absent target namespace names** of such as-yet unresolved reference **QNames** in *d* components must also be converted if is satisfied.



The above is carefully worded so that multiple *ing* of the same schema document will not constitute a violation of Schema Properties Correct 1. 2. , but applications are allowed, indeed encouraged, to avoid *ing* the same schema document more than once to forestall the necessity of establishing identity component by component.

#### 4.2.2. Including modified component definitions

In order to provide some support for evolution and versioning, it is possible to incorporate components corresponding to a schema document *with modifications*. The modifications have a pervasive impact, that is, only the redefined components are used, even when referenced from other incorporated components, whether redefined themselves or not.

A information item may contain any number of elements. Their `schemaLocation` attributes, consisting of a URI reference, identify other **schema documents**, that is information items.

The [XML Schema](#) corresponding to contains not only the components corresponding to its definition and declaration children, but also all the components of all the [XML Schemas](#) corresponding to any d schema documents. Such schema documents must either (a) have the same `targetNamespace` as the ing schema document, or (b) no `targetNamespace` at all, in which case the d schema document is converted to the ing schema document's `targetNamespace`.

The definitions within the element itself are restricted to be redefinitions of components from the d schema document, *in terms of themselves*. That is,

- Type definitions must use themselves as their base type definition;
- Attribute group definitions and model group definitions must be supersets or subsets of their original definitions, either by including exactly one reference to themselves or by containing only (possibly restricted) components which appear in a corresponding way in their d selves.

Not all the components of the d schema document need be redefined.

This mechanism is intended to provide a declarative and modular approach to schema modification, with functionality no different except in scope from what would be achieved by wholesale text copying and redefinition by editing. In particular redefining a type is not guaranteed to be side-effect free: it may have unexpected impacts on other type definitions which are based on the redefined one, even to the extent that some such definitions become ill-formed.



The pervasive impact of redefinition reinforces the need for implementations to adopt some form of lazy or 'just-in-time' approach to component construction, which is also called for in order to avoid inappropriate dependencies on the order in which definitions and references appear in (collections of) schema documents.



v1.xsd:

```
<xs:complexType name="personName">
  <xs:sequence>
    <xs:element name="title" minOccurs="0"/>
    <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="addressee" type="personName"/>
```

v2.xsd:

```
<xs:redefine schemaLocation="v1.xsd">
  <xs:complexType name="personName">
    <xs:complexContent>
      <xs:extension base="personName">
        <xs:sequence>
          <xs:element name="generation" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>

<xs:element name="author" type="personName"/>
```

The schema corresponding to `v2.xsd` has everything specified by `v1.xsd`, with the `personName` type redefined, as well as everything it specifies itself. According to this schema, elements constrained by the `personName` type may end with a `generation` element. This includes not only the `author` element, but also the `addressee` element.

### src: Redefinition Constraints and Semantics

In addition to the conditions imposed on element information items by the schema for schemas

1. If there are any element information items among the children other than then the **actual value** of the `schemaLocation` attribute must successfully resolve.
2. If the **actual value** of the `schemaLocation` attribute successfully resolves
  - A. it resolves to (a fragment of) a resource which is an XML document (see ), which in turn corresponds to a element information item in a well-formed information set, which in turn corresponds to a valid schema.
  - B. It resolves to a element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the `d` item `SII`, the valid schema `I` and the `ing` item's parent item `SII'`.

3.
  - A. `SII` has a `targetNamespace` attribute, and its **actual value** is identical to the **actual value** of the `targetNamespace` attribute of `SII'` (which must have such an attribute).
  - B. Neither `SII` nor `SII'` have a `targetNamespace` attribute.
  - C. `SII` has no `targetNamespace` attribute (but `SII'` does).
4.
  - A. or above is satisfied

the schema corresponding to `SII'` must include not only definitions or declarations corresponding to the appropriate members of its own children, but also components identical to all the **schema components** of `I`, with the exception of those explicitly redefined (see Individual Component Redefinition Corresponding to each non- member of the children of a there are one or two schema components in the `ing` schema: 1. 2. In all cases there must be a top-level definition item of the appropriate name and kind in the `d` schema document. below).

- B. above is satisfied

the schema corresponding to `SII'` must include not only definitions or declarations corresponding to the appropriate members of its own children, but also components identical to all the **schema components** of `I`, with the exception of those explicitly redefined (see Individual Component Redefinition Corresponding to each non- member of the children of a there are one or two schema components in the `ing` schema: 1. 2. In all cases there must be a top-level definition item of the appropriate name and kind in the `d` schema document. below), except that anywhere the **absent** `target namespace` name would have appeared, the **actual value** of the `targetNamespace` attribute of `SII'` is used (see in Inclusion Constraints and Semantics In addition to the conditions imposed on element information items by the schema for schemas, 1. 2. 3. It is not an error for the actual value of the `schemaLocation` attribute to fail to resolve it all, in which case no corresponding inclusion is performed. It is an error for it to resolve but the rest of clause 1 above to fail to be satisfied. Failure to resolve may well cause less than complete assessment outcomes, of course. As discussed in , QNames in XML representations may fail to resolve, rendering components incomplete and unusable because of missing subcomponents. During schema construction,



implementations must retain QName values for such references, in case an appropriately-named component becomes available to discharge the reference by the time it is actually needed. Absent target namespace names of such as-yet unresolved reference QNames in `d` components must also be converted if it is satisfied. for details).

5. Within the children, each must have a among its children and each must have a `restriction` or `extension` among its grand-children the **actual value** of whose `base` attribute must be the same as the **actual value** of its own `name` attribute plus target namespace;
6. Within the children, for each
  - A. it has a among its contents at some level the **actual value** of whose `ref` attribute is the same as the **actual value** of its own `name` attribute plus target namespace
    - i. It must have exactly one such group.
    - ii. The **actual value** of both that group's `minOccurs` and `maxOccurs` attribute must be 1 (or **absent**).
  - B. it has no such self-reference
    - i. The **actual value** of its own `name` attribute plus target namespace must successfully **resolve** to a model group definition in I.
    - ii. The of the model group definition which corresponds to it per § 3.7.2 – XML Representation of Model Group Definition Schema Components on page 59 must be a **valid restriction** of the of that model group definition in I, as defined in Particle Valid (Restriction) For a particle (call it R, for restriction) to be a valid restriction of another particle (call it B, for base) 1. 2.
7. Within the children, for each
  - A. it has an among its contents the **actual value** of whose `ref` attribute is the same as the **actual value** of its own `name` attribute plus target namespace
 

it must have exactly one such group.
  - B. it has no such self-reference
    - i. The **actual value** of its own `name` attribute plus target namespace must successfully **resolve** to an attribute group definition in I.
    - ii. The and of the attribute group definition which corresponds to it per § 3.6.2 – XML Representation of Attribute Group Definition Schema Components on page 57 must be **valid restrictions** of the and of that attribute group definition in I, as defined in , and of Derivation Valid (Restriction, Complex) If the is restriction 1. 2. 3. 4. 5. If this constraint holds of a complex type definition, it is a valid restriction of its . (where references to the base type definition are understood as references to the attribute group definition in I).



An attribute group restrictively redefined per corresponds to an attribute group whose consist all and only of those attribute uses corresponding to s explicitly present among the children of the `ing`. No inheritance from the `d` attribute group occurs. Its is similarly based purely on an explicit , if present.



**src: Individual Component Redefinition**

Corresponding to each non- member of the children of a there are one or two schema components in the ing schema:

1. The and children information items each correspond to two components:
  - A. One component which corresponds to the top-level definition item with the same name in the d schema document, as defined in § 3 – [Schema Component Details](#) on page 13, except that its name is **absent**;
  - B. One component which corresponds to the information item itself, as defined in § 3 – [Schema Component Details](#) on page 13, except that its base type definition is the component defined in 1.1 above.

This pairing ensures the coherence constraints on type definitions are respected, while at the same time achieving the desired effect, namely that references to names of redefined components in both the ing and d schema documents resolve to the redefined component as specified in 1.2 above.

2. The and children each correspond to a single component, as defined in § 3 – [Schema Component Details](#) on page 13, except that if and when a self-reference based on a `ref` attribute whose **actual value** is the same as the item's name plus target namespace is resolved, a component which corresponds to the top-level definition item of that name and the appropriate kind in I is used.

In all cases there must be a top-level definition item of the appropriate name and kind in the d schema document.



The above is carefully worded so that multiple equivalent ing of the same schema document will not constitute a violation of of Schema Properties Correct 1. 2. , but applications are allowed, indeed encouraged, to avoid ing the same schema document in the same way more than once to forestall the necessity of establishing identity component by component (although this will have to be done for the individual redefinitions themselves).

**4.2.3. References to schema components across namespaces**


As described in § 2.2 – [XML Schema Abstract Data Model](#) on page 4, every top-level schema component is associated with a target namespace (or, explicitly, with none). This section sets out the exact mechanism and syntax in the XML form of schema definition by which a reference to a foreign component is made, that is, a component with a different target namespace from that of the referring component.


Two things are required: not only a means of addressing such foreign components but also a signal to schema-aware processors that a schema document contains such references:

The element information item identifies namespaces used in external references, i.e. those whose `QName` identifies them as coming from a different namespace (or none) than the enclosing schema document's `targetNamespace`. The **actual value** of its `namespace` attribute indicates that the containing schema document may contain qualified references to schema components in that namespace (via one or more prefixes declared with namespace declarations in the normal way). If that attribute is absent, then the import allows unqualified reference to components with no target namespace. Note that components to be imported need not be in the form of a [schema document](#); the processor is free to access or construct components using means of its own choosing.

The **actual value** of the `schemaLocation`, if present, gives a hint as to where a serialization of a [schema document](#) with declarations and definitions for that namespace (or none) may be found. When no `schemaLocation` attribute is present, the schema author is leaving the identification of that schema to

the instance, application or user, via the mechanisms described below in § 4.3 – Layer 3: Schema Document Access and Web-interopability on page 112. When a `schemaLocation` is present, it must contain a single URI reference which the schema author warrants will resolve to a serialization of a [schema document](#) containing the component(s) in the `ed` namespace referred to elsewhere in the containing schema document.

 Since both the `namespace` and `schemaLocation` attribute are optional, a bare `<import/>` information item is allowed. This simply allows unqualified reference to foreign components with no target namespace without giving any hints as to where to find them.

 The same namespace may be used both for real work, and in the course of defining schema components in terms of foreign components:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:html="http://www.w3.org/1999/xhtml"
        targetNamespace="uri:mywork" xmlns:my="uri:mywork">

  <import namespace="http://www.w3.org/1999/xhtml"/>

  <annotation>
    <documentation>
      <html:p>[Some documentation for my schema]</html:p>
    </documentation>
  </annotation>

  . . .

  <complexType name="myType">
    <sequence>
      <element ref="html:p" minOccurs="0"/>
    </sequence>
    . . .
  </complexType>

  <element name="myElt" type="my:myType"/>
</schema>
```

The treatment of references as [QNames](#) implies that since (with the exception of the schema for schemas) the target namespace and the XML Schema namespace differ, without massive redeclaration of the default namespace *either* internal references to the names being defined in a schema document *or* the schema declaration and definition elements themselves must be explicitly qualified. This example takes the first option -- most other examples in this specification have taken the second.

### src: Import Constraints and Semantics

In addition to the conditions imposed on element information items by the schema for schemas

1. A. the `namespace` attribute is present
  - its [actual value](#) must not match the [actual value](#) of the enclosing `'s` `targetNamespace` attribute.
- B. the `namespace` attribute is not present
  - the enclosing must have a `targetNamespace` attribute

2. If the application schema reference strategy using the **actual values** of the `schemaLocation` and `namespace` attributes, provides a referent, as defined by Schema Document Location Strategy Given a namespace name (or none) and (optionally) a URI reference from `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`, schema-aware processors may implement any combination of the following strategies, in any order: 1. 2. 3. 4. 5. Whenever possible configuration and/or invocation options for selecting and/or ordering the implemented strategies should be provided. ,
  - A. The referent is (a fragment of) a resource which is an XML document (see ), which in turn corresponds to a element information item in a well-formed information set, which in turn corresponds to a valid schema.
  - B. The referent is a element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the item SII and the valid schema I.
3. A. there is a `namespace` attribute
 

its **actual value** must be identical to the **actual value** of the `targetNamespace` attribute of SII.
- B. there is no `namespace` attribute
 

SII must have no `targetNamespace` attribute

It is *not* an error for the application schema reference strategy to fail. It *is* an error for it to resolve but the rest of above to fail to be satisfied. Failure to find a referent may well cause less than complete **assessment** outcomes, of course.

The **schema components** (that is , , , , ) of a schema corresponding to a element information item with one or more element information items must include not only definitions or declarations corresponding to the appropriate members of its children, but also, for each of those element information items for which above is satisfied, a set of **schema components** identical to all the **schema components** of I.




The above is carefully worded so that multiple ing of the same schema document will not constitute a violation of of Schema Properties Correct 1. 2. , but applications are allowed, indeed encouraged, to avoid ing the same schema document more than once to forestall the necessity of establishing identity component by component. Given that the `schemaLocation` attribute is only a hint, it is open to applications to ignore all but the first for a given namespace, regardless of the **actual value** of `schemaLocation`, but such a strategy risks missing useful information when new `schemaLocations` are offered.


### 4.3. Layer 3: Schema Document Access and Web-interoperability

Layers 1 and 2 provide a framework for **assessment** and XML definition of schemas in a broad variety of environments. Over time, a range of standards and conventions may well evolve to support interoperability of XML Schema implementations on the World Wide Web. Layer 3 defines the minimum level of function required of all conformant processors operating on the Web: it is intended that, over time, future standards (e.g. XML Packages) for interoperability on the Web and in other environments can be introduced without the need to republish this specification.

### 4.3.1. Standards for representation of schemas and retrieval of schema documents on the Web

For interoperability, serialized [schema documents](#), like all other Web resources, may be identified by URI and retrieved using the standard mechanisms of the Web (e.g. http, https, etc.) Such documents on the Web must be part of XML documents (see ), and are represented in the standard XML schema definition form described by layer 2 (that is as element information items).

 there will often be times when a schema document will be a complete XML 1.0 document whose document element is `<?xml ...>`. There will be other occasions in which items will be contained in other documents, perhaps referenced using fragment and/or XPointer notation.

 The variations among server software and web site administration policies make it difficult to recommend any particular approach to retrieval requests intended to retrieve serialized [schema documents](#). An `Accept` header of `application/xml, text/xml; q=0.9, */*` is perhaps a reasonable starting point.

### 4.3.2. How schema definitions are located on the Web

As described in § 4.1 – [Layer 1: Summary of the Schema-validity Assessment Core](#) on page 104, processors are responsible for providing the schema components (definitions and declarations) needed for [assessment](#). This section introduces a set of normative conventions to facilitate interoperability for instance and schema documents retrieved and processed from the Web.

 As discussed above in § 4.2 – [Layer 2: Schema Documents, Namespaces and Composition](#) on page 105, other non-Web mechanisms for delivering schemas for [assessment](#) may exist, but are outside the scope of this specification.

Processors on the Web are free to undertake [assessment](#) against arbitrary schemas in any of the ways set out in § 5.2 – [Assessing Schema-Validity](#) on page 116. However, it is useful to have a common convention for determining the schema to use. Accordingly, general-purpose schema-aware processors (i.e. those not specialized to one or a fixed set of pre-determined schemas) undertaking [assessment](#) of a document on the web must behave as follows:

- unless directed otherwise by the user, [assessment](#) is undertaken on the document element information item of the specified document;
- unless directed otherwise by the user, the processor is required to construct a schema corresponding to a schema document whose `targetNamespace` is identical to the namespace name, if any, of the element information item on which [assessment](#) is undertaken.

The composition of the complete schema for use in [assessment](#) is discussed in § 4.2 – [Layer 2: Schema Documents, Namespaces and Composition](#) on page 105 above. The means used to locate appropriate schema document(s) are processor and application dependent, subject to the following requirements:

1. Schemas are represented on the Web in the form specified above in § 4.3.1 – [Standards for representation of schemas and retrieval of schema documents on the Web](#) on page 113;
2. The author of a document uses namespace declarations to indicate the intended interpretation of names appearing therein; there may or may not be a schema retrievable via the namespace name. Accordingly whether a processor's default behavior is or is not to attempt such dereferencing, it must always provide for user-directed overriding of that default.



Experience suggests that it is not in all cases safe or desirable from a performance point of view to dereference namespace names as a matter of course. User community and/or consumer/provider agreements may establish circumstances in which such dereference is a sensible default strategy: this specification allows but does not require particular communities to establish and implement such conventions. Users are always free to supply namespace names as schema location information when dereferencing *is* desired: see below.

3. On the other hand, in case a document author (human or not) created a document with a particular schema in view, and warrants that some or all of the document conforms to that schema, the `schemaLocation` and `noNamespaceSchemaLocation` attributes (in the XML Schema instance namespace, that is, `http://www.w3.org/2001/XMLSchema-instance`) (hereafter `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation`) are provided. The first records the author's warrant with pairs of URI references (one for the namespace name, and one for a hint as to the location of a schema document defining names for that namespace name). The second similarly provides a URI reference as a hint as to the location of a schema document with no `target-namespace` attribute.

Unless directed otherwise, for example by the invoking application or by command line option, processors should attempt to dereference each schema document location URI in the **actual value** of such `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes, see details below.

4. `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes can occur on any element. However, it is an error if such an attribute occurs *after* the first appearance of an element or attribute information item within an element information item initially **validated** whose namespace name it addresses. According to the rules of § 4.1 – **Layer 1: Summary of the Schema-validity Assessment Core** on page 104, the corresponding schema may be lazily assembled, but is otherwise stable throughout **assessment**. Although schema location attributes can occur on any element, and can be processed incrementally as discovered, their effect is essentially global to the **assessment**. Definitions and declarations remain in effect beyond the scope of the element on which the binding is declared.



Multiple schema bindings can be declared using a single attribute. For example consider a stylesheet:

```
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
    http://www.w3.org/1999/XSL/Transform.xsd
    http://www.w3.org/1999/xhtml
    http://www.w3.org/1999/xhtml.xsd">
```

The namespace names used in `schemaLocation` can, but need not be identical to those actually qualifying the element within whose start tag it is found or its other attributes. For example, as above, all schema location information can be declared on the document element of a document, if desired, regardless of where the namespaces are actually used.

### src: Schema Document Location Strategy

Given a namespace name (or none) and (optionally) a URI reference from `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`, schema-aware processors may implement any combination of the following strategies, in any order:

1. Do nothing, for instance because a schema containing components for the given namespace name is already known to be available, or because it is known in advance that no efforts to locate schema documents will be successful (for example in embedded systems);
2. Based on the location URI, identify an existing schema document, either as a resource which is an XML document or a element information item, in some local schema repository;
3. Based on the namespace name, identify an existing schema document, either as a resource which is an XML document or a element information item, in some local schema repository;
4. Attempt to resolve the location URI, to locate a resource on the web which is or contains or references a element;
5. Attempt to resolve the namespace name to locate such a resource.

Whenever possible configuration and/or invocation options for selecting and/or ordering the implemented strategies should be provided.

Improved or alternative conventions for Web interoperability can be standardized in the future without reopening this specification. For example, the W3C is currently considering initiatives to standardize the packaging of resources relating to particular documents and/or namespaces: this would be an addition to the mechanisms described here for layer 3. This architecture also facilitates innovation at layer 2: for example, it would be possible in the future to define an additional standard for the representation of schema components which allowed e.g. type definitions to be specified piece by piece, rather than all at once.

## 5. Schemas and Schema-validity Assessment

The architecture of schema-aware processing allows for a rich characterization of XML documents: schema validity is not a binary predicate.

This specification distinguishes between errors in schema construction and structure, on the one hand, and schema validation outcomes, on the other.

### 5.1. Errors in Schema Construction and Structure

Before [assessment](#) can be attempted, a schema is required. Special-purpose applications are free to determine a schema for use in [assessment](#) by whatever means are appropriate, but general purpose processors should implement the strategy set out in Schema Document Location Strategy Given a namespace name (or none) and (optionally) a URI reference from `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`, schema-aware processors may implement any combination of the following strategies, in any order: 1. 2. 3. 4. 5. Whenever possible configuration and/or invocation options for selecting and/or ordering the implemented strategies should be provided. , starting with the namespaces declared in the document whose [assessment](#) is being undertaken, and the [actual values](#) of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes thereof, if any, along with any other information about schema identity or schema document location provided by users in application-specific ways, if any.

It is an error if a schema and all the components which are the value of any of its properties, recursively, fail to satisfy all the relevant Constraints on Schemas set out in the last section of each of the subsections of § 3 – [Schema Component Details](#) on page 13.



If a schema is derived from one or more schema documents (that is, one or more element information items) based on the correspondence rules set out in § 3 – [Schema Component Details](#) on page 13 and § 4 – [Schemas and Namespaces: Access and Composition](#) on page 103, two additional conditions hold:

- It is an error if any such schema document would not be fully valid with respect to a schema corresponding to the [Appendix A – Schema for Schemas \(normative\)](#) on page 118, that is, following schema-validation with such a schema, the element information items would have a property with value full or partial and a property with value valid.
- It is an error if any such schema document is or contains any element information items which violate any of the relevant Schema Representation Constraints set out in [Appendix C.3 – Schema Representation Constraints](#) on page 119.

The three cases described above are the only types of error which this specification defines. With respect to the processes of the checking of schema structure and the construction of schemas corresponding to schema documents, this specification imposes no restrictions on processors after an error is detected. However [assessment](#) with respect to schema-like entities which do *not* satisfy all the above conditions is incoherent. Accordingly, conformant processors must not attempt to undertake [assessment](#) using such non-schemas.

## 5.2. Assessing Schema-Validity

With a schema which satisfies the conditions expressed in § 5.1 – [Errors in Schema Construction and Structure](#) on page 115 above, the schema-validity of an element information item can be assessed. Three primary approaches to this are possible:

1. The user or application identifies a complex type definition from among the of the schema, and appeals to Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . ();
2. The user or application identifies a element declaration from among the of the schema, checks that its and match the local name and namespace name of the item, and appeals to Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied, an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an xsi:type attribute is involved, however, takes precedence, as is made clear in . ();
3. The processor starts from Schema-Validity Assessment (Element) The schema-validity assessment of an element information item depends on its validation and the assessment of its element information item children and associated attribute information items, if any. So for an element information item's schema-validity to be assessed 1. 2. If either case of above holds, the element information item has been strictly assessed. If the item cannot be strictly assessed, because neither nor above are satisfied,



an element information item's schema validity may be laxly assessed if its context-determined declaration is not skip by validating with respect to the ur-type definition as per . In general if above holds does not, and vice versa. When an `xsi:type` attribute is involved, however, takes precedence, as is made clear in . with no stipulated declaration or definition, and either **strict** or **lax** assessment ensues, depending on whether or not the element information and the schema determine either an element declaration (by name) or a type definition (via `xsi:type`) or not.

The outcome of this effort, in any case, will be manifest in the validation attempted and validity properties on the element information item and its attributes and children, recursively, as defined by Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. and Assessment Outcome (Attribute) If the schema-validity of an attribute information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property. 1. 2. 1. 2. infoset. See for the other possible value. . It is up to applications to decide what constitutes a successful outcome.

Note that every element and attribute information item participating in the **assessment** will also have a validation context property which refers back to the element information item at which **assessment** began. This item, that is the element information item at which **assessment** began, is called the *validation root*.



This specification does not reconstruct the XML 1.0 notion of *root* in either schemas or instances. Equivalent functionality is provided for at **assessment** invocation, via above.



This specification has nothing normative to say about multiple **assessment** episodes. It should however be clear from the above that if a processor restarts **assessment** with respect to a **post-schema-validation infoset** some **post-schema-validation infoset** contributions from the previous **assessment** may be overwritten. Restarting nonetheless may be useful, particularly at a node whose validation attempted property is none, in which case there are three obvious cases in which additional useful information may result:

- **assessment** was not attempted because of a **validation** failure, but declarations and/or definitions are available for at least some of the children or attributes;
- **assessment** was not attempted because a named definition or declaration was missing, but after further effort the processor has retrieved it.
- **assessment** was not attempted because it was skipped, but the processor has at least some declarations and/or definitions available for at least some of the children or attributes.

### 5.3. Missing Sub-components

At the beginning of § 3 – **Schema Component Details** on page 13, attention is drawn to the fact that most kinds of schema components have properties which are described therein as having other components, or sets of other components, as values, but that when components are constructed on the basis of their correspondence with element information items in schema documents, such properties usually correspond to QNames, and the of such QNames may fail, resulting in one or more values of or containing **absent** where a component is mandated.

If at any time during **assessment**, an element or attribute information item is being **validated** with respect to a component of any kind any of whose properties has or contains such an **absent** value, the **validation** is modified, as following:

- In the case of attribute information items, the effect is as if of Attribute Locally Valid For an attribute information item to be locally valid with respect to an attribute declaration 1. 2. 3. 4. had failed;
- In the case of element information items, the effect is as if of Element Locally Valid (Element) For an element information item to be locally valid with respect to an element declaration 1. 2. 3. 4. 5. 6. 7. had failed;
- In the case of element information items, processors may choose to continue [assessment](#): see [lax assessment](#).

Because of the value specification for in Assessment Outcome (Element) If the schema-validity of an element information item has been assessed as per , then in the post-schema-validation infoset it has properties as follows: The nearest ancestor element information item with a property (or this element item itself if it has such a property). 1. 2. 1. 2. 3. , if this situation ever arises, the document as a whole cannot show a of full.

## 5.4. Responsibilities of Schema-aware Processors

Schema-aware processors are responsible for processing XML documents, schemas and schema documents, as appropriate given the level of conformance (as defined in § 2.4 – [Conformance](#) on page 10) they support, consistently with the conditions set out above.

## Appendix A. Schema for Schemas (normative)

The XML representation of the schema for schema documents is presented here as a normative part of the specification, and as an illustrative example of how the XML Schema language can define itself using its own constructs. The names of XML Schema language types, elements, attributes and groups defined here are evocative of their purpose, but are occasionally verbose.

There is some annotation in comments, but a fuller annotation will require the use of embedded documentation facilities or a hyperlinked external annotation for which tools are not yet readily available.

Since a schema document is an XML document, it has optional XML and doctype declarations that are provided here for completeness. The root `schema` element defines a new schema. Since this is a schema for *XML Schema: Structures*, the `targetNamespace` references the XML Schema namespace itself.



And that is the end of the schema for schema documents.

## Appendix B. References (normative)

### *XML Schemas: Datatypes*

*XML Schema Part 2: Datatypes*, Paul V. Biron and Ashok Malhotra, eds., W3C, 2 May 2001. See <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html>

### *XML Schema Requirements*

*XML Schema Requirements*, Ashok Malhotra and Murray Maloney, eds., W3C, 15 February 1999. See <http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215>

---

*XML 1.0 (Second Edition)*

*Extensible Markup Language (XML) 1.0, Second Edition*, Tim Bray et al., eds., W3C, 6 October 2000. See <http://www.w3.org/TR/2000/REC-xml-20001006>

*XML-Infoset*

*XML Information Set*, John Cowan and Richard Tobin, eds., W3C, 16 March 2001. See <http://www.w3.org/TR/2001/WD-xml-infoset-20010316/>

*XML-Namespaces*

*Namespaces in XML*, Tim Bray et al., eds., W3C, 14 January 1999. See <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

*XPointer*

*XML Pointer Language (XPointer)*, Eve Maler and Steve DeRose, eds., W3C, 8 January 2001. See <http://www.w3.org/TR/2001/WD-xptr-20010108/>

*XPath*

*XML Path Language*, James Clark and Steve DeRose, eds., W3C, 16 November 1999. See <http://www.w3.org/TR/1999/REC-xpath-19991116>

## Appendix C. Outcome Tabulations (normative)

To facilitate consistent reporting of schema errors and [validation](#) failures, this section tabulates and provides unique names for all the constraints listed in this document. Wherever such constraints have numbered parts, reports should use the name given below plus the part number, separated by a period ('.'). Thus for example `cos-ct-extends.1.2` should be used to report a violation of the of Derivation Valid (Extension) If the is extension, 1. 2. If this constraint holds of a complex type definition, it is a valid extension of its . .

### C.1. Validation Rules

### C.2. Contributions to the post-schema-validation infoset

### C.3. Schema Representation Constraints

### C.4. Schema Component Constraints

## Appendix D. Required Information Set Items and Properties (normative)

This specification requires as a precondition for [assessment](#) an information set as defined in [[XML-Infoset](#)] which supports at least the following information items and properties:

**Attribute Information Item**

local name, namespace name, normalized value

**Character Information Item**

character code

**Element Information Item**

local name, namespace name, children, attributes, in-scope namespaces or namespace attributes

**Namespace Information Item**

prefix, namespace name

In addition, infosets should support the `unparsedEntities` property of the Document Information Item. Failure to do so will mean all items of type ENTITY or ENTITIES will fail to [validate](#).

This specification does not require any destructive alterations to the input information set: all the information set contributions specified herein are additive.

This appendix is intended to satisfy the requirements for [Conformance](#) to the [XML-Infoset] specification.

## Appendix E. Schema Components Diagram (non-normative)



## Appendix F. Glossary (non-normative)

The listing below is for the benefit of readers of a printed version of this document: it collects together all the definitions which appear in the document above.

**Editor Note :**

An XSL macro is used to collect definitions from throughout the spec and gather them here for easy reference.

## Appendix G. DTD for Schemas (non-normative)

The DTD for schema documents is given below. Note there is *no* implication here that `schema` must be the root element of a document.

Although this DTD is non-normative, any XML document which is not valid per this DTD, given redefinitions in its internal subset of the 'p' and 's' parameter entities below appropriate to its namespace declaration of the XML Schema namespace, is almost certainly not a valid schema document, with the exception of documents with multiple namespace prefixes for the XML Schema namespace itself. Accordingly authoring XML Schema documents using this DTD and DTD-based authoring tools, and specifying it as the DOCTYPE of documents intended to be XML Schema documents and validating them with a validating XML parser, are sensible development strategies which users are encouraged to adopt until XML Schema-based authoring tools and validators are more widely available.

## Appendix H. Analysis of the Unique Particle Attribution Constraint (non-normative)

A specification of the import of Unique Particle Attribution A content model must be formed such that during validation of an element information item sequence, the particle component contained directly, indirectly or implicitly therein with which to attempt to validate each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence. This constraint reconstructs for XML Schema the equivalent constraints of and SGML. Given the presence of element substitution groups and wildcards, the concise expression of this constraint is difficult, see for further discussion. Since this constraint is expressed at the component level, it applies to content models whose origins (e.g. via type derivation and references to named model groups) are no longer evident. So particles at different points in the content model are always distinct from one another, even if they originated from the same named model group. which does not appeal to a processing model is difficult. What follows is intended as guidance, without claiming to be complete.

Two non-group particles *overlap* if

- They are both element declaration particles whose declarations have the same and .
- or
- They are both element declaration particles one of whose and are the same as those of an element declaration in the other's [substitution group](#).
- or
- They are both wildcards, and the intensional intersection of their s as defined in Attribute Wildcard Intersection For a wildcard's value to be the intensional intersection of two other such values (call them O1 and O2): 1. 2. 3. 4. 5. 6. In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required. is not the empty set.
- or
- One is a wildcard and the other an element declaration, and the of any member of its [substitution group](#) is [valid](#) with respect to the of the wildcard.

A content model will violate the unique attribution constraint if it contains two particles which [overlap](#) and which either

- are both in the of a choice or all group
- or
- may [validate](#) adjacent information items and the first has less than .

Two particles may [validate](#) adjacent information items if they are separated by at most epsilon transitions in the most obvious transcription of a content model into a finite-state automaton.

A precise formulation of this constraint can also be offered in terms of operations on finite-state automaton: transcribe the content model into an automaton in the usual way using epsilon transitions for optionality and unbounded maxOccurs, unfolding other numeric occurrence ranges and treating the heads of substitution groups as if they were choices over all elements in the group, *but* using not element QNames as transition

labels, but rather pairs of element QNames and positions in the model. Determinize this automaton, treating wildcard transitions as opaque. Now replace all QName+position transition labels with the element QNames alone. If the result has any states with two or more identical-QName-labeled transitions from it, or a QName-labeled transition and a wildcard transition which subsumes it, or two wildcard transitions whose intentional intersection is non-empty, the model does not satisfy the Unique Attribution constraint.

## Appendix I. References (non-normative)

### DCD

*Document Content Description for XML (DCD)*, Tim Bray et al., eds., W3C, 10 August 1998. See <http://www.w3.org/TR/1998/NOTE-dcd-19980731>

### DDML

*Document Definition Markup Language*, Ronald Bourret, John Cowan, Ingo Macherius, Simon St. Laurent, eds., W3C, 19 January 1999. See <http://www.w3.org/TR/1999/NOTE-ddml-19990119>

### XML Schema: Primer

*XML Schema Part 0: Primer*, David C. Fallside, ed., W3C, 2 May 2001. See <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/primer.html>

### SOX

*Schema for Object-oriented XML*, Andrew Davidson et al., eds., W3C, 1998. See <http://www.w3.org/1999/07/NOTE-SOX-19990730/>

### SOX-2

*Schema for Object-oriented XML*, Version 2.0, Andrew Davidson, et al., W3C, 30 July 1999. See <http://www.w3.org/TR/NOTE-SOX/>

### XDR

*XML-Data Reduced*, Charles Frankston and Henry S. Thompson, 3 July 1998. See <http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm>

### XML-Data

*XML-Data*, Andrew Layman et al., W3C, 05 January 1998. See <http://www.w3.org/TR/1998/NOTE-XML-data-0105/>

## Appendix J. Acknowledgements (non-normative)

The following contributed material to the first edition of this specification:

- David Fallside, IBM
- Scott Lawrence, Agranat Systems
- Andrew Layman, Microsoft
- Eve L. Maler, Sun Microsystems
- Asir S. Vedamuthu, webMethods, Inc

The editors acknowledge the members of the XML Schema Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the

process or content of creating this document. The Working Group is particularly grateful to Lotus Development Corp. and IBM for providing teleconferencing facilities.

At the time the first edition of this specification was published, the members of the XML Schema Working Group were:

Jim Barnette, Defense Information Systems Agency (DISA); Paul V. Biron, Health Level Seven; Don Box, DevelopMentor; Allen Brown, Microsoft; Lee Buck, TIBCO Extensibility; Charles E. Campbell, Informix; Wayne Carr, Intel; Peter Chen, Bootstrap Alliance and LSU; David Cleary, Progress Software; Dan Connolly, W3C (staff contact); Ugo Corda, Xerox; Roger L. Costello, MITRE; Haavard Danielson, Progress Software; Josef Dietl, Mozquito Technologies; David Ezell, Hewlett-Packard Company ; Alexander Falk, Altova GmbH; David Fallside, IBM; Dan Fox, Defense Logistics Information Service (DLIS); Matthew Fuchs, Commerce One; Andrew Goodchild, Distributed Systems Technology Centre (DSTC Pty Ltd); Paul Grosso, Arbortext, Inc; Martin Gudgin, DevelopMentor; Dave Hollander, Contivo, Inc (co-chair); Mary Holstege, Invited Expert; Jane Hunter, Distributed Systems Technology Centre (DSTC Pty Ltd); Rick Jelliffe, Academia Sinica; Simon Johnston, Rational Software; Bob Lojek, Mozquito Technologies; Ashok Malhotra, Microsoft; Lisa Martin, IBM; Noah Mendelsohn, Lotus Development Corporation; Adrian Michel, Commerce One; Alex Milowski, Invited Expert; Don Mullen, TIBCO Extensibility; Dave Peterson, Graphic Communications Association; Jonathan Robie, Software AG; Eric Sedlar, Oracle Corp.; C. M. Sperberg-McQueen, W3C (co-chair); Bob Streich, Calico Commerce; William K. Stumbo, Xerox; Henry S. Thompson, University of Edinburgh; Mark Tucker, Health Level Seven; Asir S. Vedamuthu, webMethods, Inc; Priscilla Walmsley, XMLSolutions; Norm Walsh, Sun Microsystems; Aki Yoshida, SAP AG; Kongyi Zhou, Oracle Corp.

The XML Schema Working Group has benefited in its work from the participation and contributions of a number of people not currently members of the Working Group, including in particular those named below. Affiliations given are those current at the time of their work with the WG.

Paula Angerstein, Vignette Corporation; David Beech, Oracle Corp.; Gabe Begeg-Dov, Rogue Wave Software; Greg Bumgardner, Rogue Wave Software; Dean Burson, Lotus Development Corporation; Mike Cokus, MITRE; Andrew Eisenberg, Progress Software; Rob Ellman, Calico Commerce; George Feinberg, Object Design; Charles Frankston, Microsoft; Ernesto Guerrieri, Inso; Michael Hyman, Microsoft; Renato Iannella, Distributed Systems Technology Centre (DSTC Pty Ltd); Dianne Kennedy, Graphic Communications Association; Janet Koenig, Sun Microsystems; Setrag Khoshafian, Technology Deployment International (TDI); Ara Kullukian, Technology Deployment International (TDI); Andrew Layman, Microsoft; Dmitry Lenkov, Hewlett-Packard Company; John McCarthy, Lawrence Berkeley National Laboratory; Murata Makoto, Xerox; Eve Maler, Sun Microsystems; Murray Maloney, Muzmo Communication, acting for Commerce One; Chris Olds, Wall Data; Frank Olken, Lawrence Berkeley National Laboratory; Shriram Revankar, Xerox; Mark Reinhold, Sun Microsystems; John C. Schneider, MITRE; Lew Shannon, NCR; William Shea, Merrill Lynch; Ralph Swick, W3C; Tony Stewart, Rivcom; Matt Timmermans, Microstar; Jim Trezzo, Oracle Corp.; Steph Tryphonas, Microstar

The lists given above pertain to the first edition. At the time work on this second edition was completed, the membership of the Working Group was:

Leonid Arbousov, Sun Microsystems; Jim Barnette, Defense Information Systems Agency (DISA); Paul V. Biron, Health Level Seven; Allen Brown, Microsoft; Charles E. Campbell, Invited expert; Peter Chen, Invited expert; Tony Cincotta, NIST; David Ezell, National Association of Convenience Stores; Matthew Fuchs, Invited expert; Sandy Gao, IBM; Andrew Goodchild, Distributed Systems Technology Centre (DSTC Pty Ltd); Xan Gregg, Invited expert; Mary Holstege, Mark Logic; Mario Jeckle, DaimlerChrysler; Marcel Jemio, Data Interchange Standards Association; Kohsuke Kawaguchi, Sun Microsystems; Ashok Malhotra, Invited expert; Lisa Martin, IBM; Jim Melton, Oracle Corp; Noah Mendelsohn, IBM; Dave Peterson, Invited expert; Anli Shundi, TIBCO Extensibility; C. M. Sperberg-McQueen, W3C (co-chair);



---

Hoylen Sue, Distributed Systems Technology Centre (DSTC Pty Ltd); Henry S. Thompson, University of Edinburgh; Asir S. Vedamuthu, webMethods, Inc; Priscilla Walmsley, Invited expert; Kongyi Zhou, Oracle Corp.

We note with sadness the accidental death of Mario Jeckle shortly after the completion of work on this document. In addition to those named above, several people served on the Working Group during the development of this second edition:

Oriol Carbo, University of Edinburgh; Tyng-Ruey Chuang, Academia Sinica; Joey Coyle, Health Level 7; Tim Ewald, DevelopMentor; Nelson Hung, Corel; Melanie Kudela, Uniform Code Council; Matthew MacKenzie, XML Global; Cliff Schmidt, Microsoft; John Stanton, Defense Information Systems Agency; John Tebbutt, NIST; Ross Thompson, Contivo; Scott Vorthmann, TIBCO Extensibility