**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](http://www.w3.org/2004/02/Process-20040205/tr.html#RecsW3C), 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](http://www.w3.org/XML/Schema) as part of the W3C [XML Activity](http://www.w3.org/XML/Activity). The goals of the XML Schema language are discussed in the [XML Schema Requirements](http://www.w3.org/TR/NOTE-xml-schema-req) document. The authors of this document are the members of the XML Schema Working Group. Different parts of this specification have different editors.

This version of this document incorporates some editorial changes from earlier versions.

This document was produced under the [24 January 2002 Current Patent Practice (CPP)](http://www.w3.org/TR/2002/NOTE-patent-practice-20020124) as amended by the [W3C Patent Policy Transition Procedure](http://www.w3.org/2004/02/05-pp-transition). The Working Group maintains a [public list of patent disclosures](http://www.w3.org/2002/11/xml-schema-IPR-statements.html) 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](http://www.w3.org/Consortium/Patent-Policy-20040205/#sec-Disclosure).

Please report errors in this document to [www-xml-schema-comments@w3.org](mailto:www-xml-schema-comments@w3.org) ([archive](http://lists.w3.org/Archives/Public/www-xml-schema-comments/)). The list of known errors in this specification is available at <http://www.w3.org/2001/05/xmlschema-errata>.

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>.

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

This second edition is not a new version, it merely incorporates the changes dictated by the corrections to errors found in the [first edition](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/) 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](http://lists.w3.org/Archives/Public/www-xml-schema-comments/)).

**Note**

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)

**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.

# Introduction

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

Chapter 2 presents a [Conceptual Framework](#concepts) 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, [Schema Component Details](#components), 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 [Schemas and Namespaces: Access and Composition](#composition), 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 [Schemas and Schema-validity Assessment](#conformance), including the overall approach to schema-validity assessment of documents, and responsibilities of schema-aware processors.

The normative appendices include a [Schema for Schemas (normative)](#normative-schemaSchema) for the XML representation of schemas and [References (normative)](#normative-references).

The non-normative appendices include the [DTD for Schemas (non-normative)](#nonnormative-schemaDTD) and a [Glossary (non-normative)](#normative-glossary).

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]](#bib-expo) is a much better starting point than this document.

## 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.

## Dependencies on Other Specifications

The definition of XML Schema: Structures depends on the following specifications: [[XML-Infoset]](#ref-xmlinfo), [[XML-Namespaces]](#ref-xml-namespaces), [[XPath]](#bib-xpath), and [[XML Schemas: Datatypes]](#ref-xsp2).

See [Required Information Set Items and Properties (normative)](#infoset) for a tabulation of the information items and properties specified in [[XML-Infoset]](#ref-xmlinfo) which this specification requires as a precondition to schema-aware processing.

## 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 [Definition:] 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](#key-sampledef).

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

**Example**

<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:

**Schema Component:** [Example](#intro-terminology)

{example property} 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 [{example property}](#xmpl-prop).

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]](#ref-xsp2), 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.

**Note**

The illustrations are derived automatically from the [Schema for Schemas (normative)](#normative-schemaSchema). In the case of apparent conflict, the [Schema for Schemas (normative)](#normative-schemaSchema) takes precedence, as it, together with the [Schema Representation Constraints](#gloss-src), provide the normative statement of the form of XML representations.

**XML Representation Summary:** example **Element Information Item**

[Example](#intro-terminology) **Schema Component**

**Property Representation**

{example property} Description of what the property corresponds to, e.g. the value of the size [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) attribute

References to elements in the text are links to the relevant illustration as exemplified above, set off with angle brackets, for instance [<example>](#example).

References to properties of information items as defined in [[XML-Infoset]](#ref-xmlinfo) are notated as links to the relevant section thereof, set off with square brackets, for example [children](http://www.w3.org/TR/xml-infoset/#infoitem.element).children.

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

{new property} 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 [{new property}](#ex-foo).

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

**Note**

General comments directed to all readers.

Following [[XML 1.0 (Second Edition)]](#ref-xml), 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 [Schemas and Schema-validity Assessment](#conformance)) which is considerably more complex than that of [[XML 1.0 (Second Edition)]](#ref-xml).

# Conceptual Framework

This chapter gives an overview of XML Schema: Structures at the level of its abstract data model. [Schema Component Details](#components) 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]](#bib-expo) for a tutorial introduction, and only then consult the sub-sections of [Schema Component Details](#components) named XML Representation of ... for the details.

## 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]](#ref-xmlinfo)), 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. [Definition:] 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:

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;

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, [Definition:] the word valid and its derivatives are used to refer to .

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

## XML Schema Abstract Data Model

This specification builds on [[XML 1.0 (Second Edition)]](#ref-xml) and [[XML-Namespaces]](#ref-xml-namespaces). The concepts and definitions used herein regarding XML are framed at the abstract level of [information items](http://www.w3.org/TR/xml-infoset/#infoitem) as defined in [[XML-Infoset]](#ref-xmlinfo). By definition, this use of the infoset provides a priori guarantees of [well-formedness](http://www.w3.org/TR/REC-xml#sec-well-formed) (as defined in [[XML 1.0 (Second Edition)]](#ref-xml)) and [namespace conformance](http://www.w3.org/TR/REC-xml-names/#Conformance) (as defined in [[XML-Namespaces]](#ref-xml-namespaces)) for all candidates for [assessment](#key-va) and for all [schema documents](#key-schemaDoc).

Just as [[XML 1.0 (Second Edition)]](#ref-xml) and [[XML-Namespaces]](#ref-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 interoperation and sharing of schema information, a normative XML interchange format for schemas is provided.

[Definition:] Schema component is the generic term for the building blocks that comprise the abstract data model of the schema. [Definition:] An XML Schema is a set of [schema components](#key-component). 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](#key-vn), [Definition:] declaration components are associated by (qualified) name to information items being [validated](#key-vn).

On the other hand, [Definition:] definition components define internal schema components that can be used in other schema components.

[Definition:] Declarations and definitions may have and be identified by names, which are NCNames as defined by [[XML-Namespaces]](#ref-xml-namespaces).

[Definition:] Several kinds of component have a target namespace, which is either [absent](#key-null) or a namespace name, also as defined by [[XML-Namespaces]](#ref-xml-namespaces). The [target namespace](#key-targetNS) 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](#key-vn).

**Note**

At the abstract level, there is no requirement that the components of a schema share a [target namespace](#key-targetNS). Any schema for use in [assessment](#key-va) of documents containing names from more than one namespace will of necessity include components with different [target namespaces](#key-targetNS). 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](#key-vn), defined in detail in [Schema Component Details](#components), is a relation between information items and schema components. For example, an attribute information item may [validate](#key-vn) with respect to an attribute declaration, a list of element information items may [validate](#key-vn) 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](#key-vn).

### Type Definition Components

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

[Definition:] 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.

#### Type Definition Hierarchy

[Definition:] Except for a distinguished [ur-type definition](#key-urType), every [type definition](#key-typeDefn) is, by construction, either a [restriction](#key-typeRestriction) or an [extension](#key-typeExtension) of some other type definition. The graph of these relationships forms a tree known as the Type Definition Hierarchy.

[Definition:] 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](#key-typeRestriction) of the definition of another type, B, are always members of type B as well.

[Definition:] 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.

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

[Definition:] A type definition used as the basis for an [extension](#key-typeExtension) or [restriction](#key-typeRestriction) is known as the base type definition of that definition.

#### 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](#key-nv) 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]](#ref-xsp2)) or user-defined, is a [restriction](#key-typeRestriction) of some particular simple [base type definition](#key-baseTypeDefinition). For the built-in primitive type definitions, this is [Definition:] the simple ur-type definition, a special restriction of the [ur-type definition](#key-urType), whose name is anySimpleType in the XML Schema namespace. The [simple ur-type definition](#key-simpleUrType) 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](#key-simpleUrType). 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](#key-simpleUrType), checking identity constraints involving such items.

**Note**

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](#key-simpleUrType).

For detailed information on simple type definitions, see [Simple Type Definitions](#Simple_Type_Definitions) and [[XML Schemas: Datatypes]](#ref-xsp2). The latter also defines an extensive inventory of pre-defined simple types.

#### Complex Type Definition

A complex type definition is a set of attribute declarations and a content type, applicable to the [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) and attributes and [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of an element information item respectively. The content type may require the children of an element information item respectively. The content type may require the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) 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.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](#key-urType) is either or

a restriction of a complex [base type definition](#key-baseTypeDefinition)

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

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.

**Note**

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 [Complex Type Definitions](#Complex_Type_Definitions).

### 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.

#### 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 declarations contribute to [validation](#key-vn) as part of model group [validation](#key-vn), 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](#key-vn).

For detailed information on element declarations, see [Element Declarations](#cElement_Declarations).

#### 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.

[Definition:] 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](#key-vn) 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 [Element Declarations](#cElement_Declarations)).

#### 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](#key-vn) as part of complex type definition [validation](#key-vn), 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 [Attribute Declarations](#cAttribute_Declarations).

#### 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](#key-vn) 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 [Notation Declarations](#cNotation_Declarations).

### 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.

#### 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 [Model Groups](#Model_Groups).

#### 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](#key-vn) as part of complex type definition [validation](#key-vn), when they allow anywhere from zero to many element information items or sequences thereof, depending on their contents and occurrence constraints.

[Definition:] A particle can be used in a complex type definition to constrain the [validation](#key-vn) of the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of an element information item; such a particle is called a children of an element information item; such a particle is called a content model.

**Note**

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

For detailed information on particles, see [Particles](#cParticles).

#### 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.

#### 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 [Wildcards](#Wildcards).

### 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]](#bib-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](#key-vn) reference, within a specified scope. An element information item is only [valid](#key-vn) 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 [Identity-constraint Definitions](#cIdentity-constraint_Definitions).

### 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.

#### 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 [Model Group Definitions](#cModel_Group_Definitions).

#### 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 [Attribute Group Definitions](#cAttribute_Group_Definitions).

### 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 [Annotations](#cAnnotations).

## Constraints and Validation Rules

The [[XML 1.0 (Second Edition)]](#ref-xml) 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](#key-vn), 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](#key-vn) of information items:

**Schema Component Constraint**

[Definition:] 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 [Schema Component Details](#components) and tabulated in [Schema Component Constraints](#outcome-cos).

**Schema Representation Constraint**

[Definition:] Constraints on the representation of schema components in XML beyond those which are expressed in [Schema for Schemas (normative)](#normative-schemaSchema). Located in the third sub-section of the per-component sections of [Schema Component Details](#components) and tabulated in [Schema Representation Constraints](#outcome-src).

**Validation Rules**

[Definition:] Contributions to [validation](#key-vn) associated with schema components. Located in the fourth sub-section of the per-component sections of [Schema Component Details](#components) and tabulated in [Validation Rules](#validation_failures).

**Schema Information Set Contribution**

[Definition:] Augmentations to [post-schema-validation infoset](#key-psvi)s expressed by schema components, which follow as a consequence of [validation](#key-vn) and/or [assessment](#key-va). Located in the fifth sub-section of the per-component sections of [Schema Component Details](#components) and tabulated in [Contributions to the post-schema-validation infoset](#PSVI_contributions).

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.

## 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.

[Definition:] Minimally conforming processors must completely and correctly implement the [Schema Component Constraints](#gloss-cos), [Validation Rules](#gloss-cvc), and [Schema Information Set Contributions](#gloss-sic) contained in this specification.

[Definition:] [Minimally conforming](#key-minimallyConforming) processors which accept schemas represented in the form of XML documents as described in [Layer 2: Schema Documents, Namespaces and Composition](#layer2) 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](#gloss-src) in this specification, and must adhere exactly to the specifications in [Schema Component Details](#components) for mapping the contents of such documents to [schema components](#key-component) for use in [validation](#key-vn) and [assessment](#key-va).

**Note**

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](#key-minimallyConforming) but not necessarily in [conformance to the XML Representation of Schemas](#key-interchange).

[Definition:] Fully conforming processors are network-enabled processors which are not only both [minimally conforming](#key-minimallyConforming) and [in conformance to the XML Representation of Schemas](#key-interchange), but which additionally must be capable of accessing schema documents from the World Wide Web according to [Representation of Schemas on the World Wide Web](#web-representation) and [How schema definitions are located on the Web](#schema-loc). .

**Note**

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 [Schemas and Namespaces: Access and Composition](#composition) for a more detailed explanation of the mechanisms supporting these levels of conformance.

## Names and Symbol Spaces

As discussed in [XML Schema Abstract Data Model](#concepts-data-model), most schema components (may) have [names](#key-compName). 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](#key-targetNS).

Therefore [Definition:] 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](http://www.w3.org/TR/REC-xml-names/#ns-breakdown) introduced in [[XML-Namespaces]](#ref-xml-namespaces). There is a single distinct symbol space within a given [target namespace](#key-targetNS) for each kind of definition and declaration component identified in [XML Schema Abstract Data Model](#concepts-data-model), 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.

## 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](#xsi.type), [Attribute Declaration for the 'nil' attribute](#xsi.nil), [Attribute Declaration for the 'schemaLocation' attribute](#xsi.schemaLocation) and [Attribute Declaration for the 'noNamespaceSchemaLocation' attribute](#xsi.noNamespaceSchemaLocation).

### xsi:type

The [Simple Type Definition](#Simple_Type_Definition) or [Complex Type Definition](#Complex_Type_Definition) used in [validation](#key-vn) 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](#gloss-QName); see [QName Interpretation](#src-qname) for the means by which the [QName](#gloss-QName) is associated with a type definition.

### xsi:nil

XML Schema: Structures introduces a mechanism for signaling that an element should be accepted as [valid](#key-vn) when it has no content despite a content type which does not require or even necessarily allow empty content. An element may be [valid](#key-vn) 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.

### 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](#key-va). See [How schema definitions are located on the Web](#schema-loc) for details on the use of these attributes.

## 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 [Inclusion Constraints and Semantics](#src-include)), conforming to the specifications in [Layer 2: Schema Documents, Namespaces and Composition](#layer2). For more information on the representation and use of schema documents on the World Wide Web see [Standards for representation of schemas and retrieval of schema documents on the Web](#schema-repr) and [How schema definitions are located on the Web](#schema-loc).

# Schema Component Details

## Introduction

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

properties: their values and significance

XML representation and the mapping to properties

constraints on representation

validation rules

[post-schema-validation infoset](#key-psvi) contributions

constraints on the components themselves

### 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](#key-symbolSpace) 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.

**Note**

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".

[Definition:] 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](#key-null) 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](#key-null). 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](http://www.w3.org/TR/REC-xml#charsets) in [[XML 1.0 (Second Edition)]](#ref-xml).

### 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. [Definition:] A document in this form (i.e. a [<schema>](#schema) 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 [Schema for Schemas (normative)](#normative-schemaSchema) and [DTD for Schemas (non-normative)](#nonnormative-schemaDTD)) 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](http://www.w3.org/TR/xml-infoset/#infoitem.element) must be 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](#key-schemaDoc) will be using an XML parser, this is not required: any mechanism which constructs conformant infosets as defined in [[XML-Infoset]](#ref-xmlinfo) is a possible starting point.

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

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;

All of them allow an [<annotation>](#annotation) as their first child, for human-readable documentation and/or machine-targeted information.

### 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 [Schema for Schemas (normative)](#normative-schemaSchema).

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](http://www.w3.org/TR/xml-infoset/#infoitem.element) of an element information item. Since schema documents are constrained by the attributes of an element information item. Since schema documents are constrained by the [Schema for Schemas (normative)](#normative-schemaSchema), there is always a simple type definition associated with any such attribute information item. [Definition:] 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](#key-nv). 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](#key-va).

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 [Missing Sub-components](#conformance-missing): 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](#key-schemaDoc). By the time the component corresponding to an XML representation which contains a forward reference is actually needed for [validation](#key-vn) an appropriately-named component may have become available to discharge the reference: see [Schemas and Namespaces: Access and Composition](#composition) for details.

### White Space Normalization during Validation

Throughout this specification, [Definition:] the initial value of some attribute information item is the value of the [normalized value](http://www.w3.org/TR/xml-infoset/#infoitem.attribute) property of that item. Similarly, 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](http://www.w3.org/TR/xml-infoset/#infoitem.character) of each character information item in the character code of each character information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of that element information itemchildren 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](#key-vn) purposes.

[Definition:] The normalized value of an element or attribute information item is an [initial value](#key-iv) whose white space, if any, has been normalized according to the value of the [whiteSpace facet](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#rf-whiteSpace) of the simple type definition used in its [validation](#key-vn):

**preserve**

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

**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](#key-vn) is the [simple ur-type definition](#key-simpleUrType), the [normalized value](#key-nv) 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](#cvc-attribute) ([Element Locally Valid (Type)](#cvc-type) ([Element Locally Valid (Complex Type)](#cvc-complex-type) (

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](http://www.w3.org/TR/REC-xml#AVNormalize) in [[XML 1.0 (Second Edition)]](#ref-xml) for the precedent for replace and collapse for attributes. Extending this processing to element content is necessary to ensure a consistent [validation](#key-vn) 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](http://www.w3.org/TR/xml-infoset/#infoitem.attribute) 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.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.

**Note**

Even when DTD-based information has been appealed to, and [Attribute Value Normalization](http://www.w3.org/TR/REC-xml#AVNormalize) has taken place, the above definition of [normalized value](#key-nv) 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 value](#key-iv)s.

## Attribute Declarations

Attribute declarations provide for:

Local [validation](#key-vn) of attribute information item values using a simple type definition;

Specifying default or fixed values for attribute information items.

**Example**

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

The XML representation of an attribute declaration.

### The Attribute Declaration Schema Component

The attribute declaration schema component has the following properties:

**Schema Component:** [Attribute Declaration](#Attribute_Declaration)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{type definition} A simple type definition.

{scope} Optional. Either global or a complex type definition.

{value constraint} Optional. A pair consisting of a value and one of default, fixed.

{annotation} Optional. An annotation.

The [{name}](#a-name) property must match the local part of the names of attributes being [validated](#key-vn).

The value of the attribute must conform to the supplied [{type definition}](#a-simple_type_definition).

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

A [{scope}](#a-scope) 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 [{scope}](#a-scope) property. This property is [absent](#key-null) 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.

[{value constraint}](#a-value_constraint) 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](#key-psvi), 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 [Annotations](#cAnnotations) for information on the role of the [{annotation}](#a-annotation) property.

**Note**

A more complete and formal presentation of the semantics of [{name}](#a-name), [{target namespace}](#a-target_namespace) and [{value constraint}](#a-value_constraint) is provided in conjunction with other aspects of complex type [validation](#key-vn) (see [Element Locally Valid (Complex Type)](#cvc-complex-type).)

[[XML-Infoset]](#ref-xmlinfo) distinguishes attributes with names such as xmlns or xmlns:xsl from ordinary attributes, identifying them as [namespace attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element). Accordingly, it is unnecessary and in fact not possible for schemas to contain attribute declarations corresponding to such namespace declarations, see 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](#no-xmlns). No means is provided in this specification to supply a default value for a namespace declaration.

### XML Representation of Attribute Declaration Schema Components

The XML representation for an attribute declaration schema component is an [<attribute>](#attribute) 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:

**XML Representation Summary:** attribute **Element Information Item**

If the [<attribute>](#attribute) element information item has [<schema>](#schema) as its parent, the corresponding schema component is as follows:

[Attribute Declaration](#Attribute_Declaration_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent [<schema>](#schema) element information item, or [absent](#key-null) if there is none.

{type definition} The simple type definition corresponding to the [<simpleType>](#simpleType) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise the simple type definition children, if present, otherwise the simple type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the type [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise the attribute, if present, otherwise the [simple ur-type definition](#simple-ur-type-itself).

{scope} global.

{value constraint} If there is a default or a fixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), then a pair consisting of the attribute, then a pair consisting of the [actual value](#key-vv) (with respect to the [{type definition}](#a-simple_type_definition)) of that [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) and either attribute and either default or fixed, as appropriate, otherwise [absent](#key-null).

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

otherwise if the [<attribute>](#attribute) element information item has [<complexType>](#complexType) or [<attributeGroup>](#attributeGroup) as an ancestor and the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is absent, it corresponds to an attribute use with properties as follows (unless 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):

[Attribute Use](#AU_details) **Schema Component**

**Property Representation**

{required} true if the use [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present with attribute is present with [actual value](#key-vv) required, otherwise false.

{attribute declaration} See the Attribute Declaration mapping immediately below.

{value constraint} If there is a default or a fixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), then a pair consisting of the attribute, then a pair consisting of the [actual value](#key-vv) (with respect to the [{type definition}](#a-simple_type_definition) of the [{attribute declaration}](#attribute)) of that [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) and either attribute and either default or fixed, as appropriate, otherwise [absent](#key-null).

[Attribute Declaration](#Attribute_Declaration_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} If form is present and its [actual value](#key-vv) is qualified, or if form is absent and the [actual value](#key-vv) of attributeFormDefault on the [<schema>](#schema) ancestor is qualified, then the [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent [<schema>](#schema) element information item, or [absent](#key-null) if there is none, otherwise [absent](#key-null).

{type definition} The simple type definition corresponding to the [<simpleType>](#simpleType) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise the simple type definition children, if present, otherwise the simple type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the type [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise the attribute, if present, otherwise the [simple ur-type definition](#simple-ur-type-itself).

{scope} If the [<attribute>](#attribute) element information item has [<complexType>](#complexType) as an ancestor, the complex definition corresponding to that item, otherwise (the [<attribute>](#attribute) element information item is within an [<attributeGroup>](#attributeGroup) definition), [absent](#key-null).

{value constraint} [absent](#key-null).

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

otherwise (the [<attribute>](#attribute) element information item has [<complexType>](#complexType) or [<attributeGroup>](#attributeGroup) as an ancestor and the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present), it corresponds to an attribute use with properties as follows (unless 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):

[Attribute Use](#AU_details) **Schema Component**

**Property Representation**

{required} true if the use [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present with attribute is present with [actual value](#key-vv) required, otherwise false.

{attribute declaration} The (top-level) attribute declaration [resolved](#src-resolve) to by the [actual value](#key-vv) of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{value constraint} If there is a default or a fixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), then a pair consisting of the attribute, then a pair consisting of the [actual value](#key-vv) (with respect to the [{type definition}](#a-simple_type_definition) of the [{attribute declaration}](#attribute)) of that [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) and either attribute and either default or fixed, as appropriate, otherwise [absent](#key-null).

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 [<simpleType>](#simpleType) is provided inline.

The default when no simple type definition is referenced or provided is the [simple ur-type definition](#key-simpleUrType), which imposes no constraints at all.

Attribute information items [validated](#key-vn) by a top-level declaration must be qualified with the [{target namespace}](#a-target_namespace) of that declaration (if this is [absent](#key-null), the item must be unqualified). Control over whether attribute information items [validated](#key-vn) by a local declaration must be similarly qualified or not is provided by the form [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), whose default is provided by the attribute, whose default is provided by the attributeFormDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) on the enclosing attribute on the enclosing [<schema>](#schema), via its determination of [{target namespace}](#a-target_namespace).

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

### Constraints on XML Representations of Attribute Declarations

### Attribute Declaration Validation Rules

### Attribute Declaration Information Set Contributions

### Constraints on Attribute Declaration Schema Components

All attribute declarations (see [Attribute Declarations](#cAttribute_Declarations)) must satisfy the following constraints.

### Built-in Attribute Declarations

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

## Element Declarations

Element declarations provide for:

Local [validation](#key-vn) 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](#key-equivalenceClass).

**Example**

<xs:element name="PurchaseOrder" type="PurchaseOrderType"/>  
  
<xs:element name="gift">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="birthday" type="xs:date"/>  
 <xs:element ref="PurchaseOrder"/>  
 </xs:sequence>  
 </xs:complexType>  
</xs:element>

XML representations of several different types of element declaration

### The Element Declaration Schema Component

The element declaration schema component has the following properties:

**Schema Component:** [Element Declaration](#Element_Declaration)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{type definition} Either a simple type definition or a complex type definition.

{scope} Optional. Either global or a complex type definition.

{value constraint} Optional. A pair consisting of a value and one of default, fixed.

{nillable} A boolean.

{identity-constraint definitions} A set of constraint definitions.

{substitution group affiliation} Optional. A top-level element definition.

{substitution group exclusions} A subset of {extension, restriction}.

{disallowed substitutions} A subset of {substitution, extension, restriction}.

{abstract} A boolean.

{annotation} Optional. An annotation.

The [{name}](#e-name) property must match the local part of the names of element information items being [validated](#key-vn).

A [{scope}](#e-scope) 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 [{scope}](#e-scope) property. This property is [absent](#key-null) 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](#key-null) value of the [{target namespace}](#e-target_namespace) property provides for [validation](#key-vn) of namespace-qualified element information items. [Absent](#key-null) values of [{target namespace}](#e-target_namespace) [validate](#key-vn) unqualified items.

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

If [{nillable}](#nillable) is true, then an element may also be [valid](#key-vn) if it carries the namespace qualified attribute with [local name](http://www.w3.org/TR/xml-infoset/#infoitem.attribute) local name nil from namespace http://www.w3.org/2001/XMLSchema-instance and value true (see [xsi:nil](#xsi_nil)) even if it has no text or element content despite a [{content type}](#content_type) which would otherwise require content. Formal details of element [validation](#key-vn) are described in [Element Locally Valid (Element)](#cvc-elt).

[{value constraint}](#e-value_constraint) establishes a default or fixed value for an element. If default is specified, and if the element being [validated](#key-vn) is empty, then the canonical form of the supplied constraint value becomes the [{schema normalized value}](#e-schema_normalized_value) of the [validated](#key-vn) element in the [post-schema-validation infoset](#key-psvi). 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.

**Note**

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 [{value constraint}](#e-value_constraint) whose simple type definition includes the empty string in its lexical space will nonetheless never receive that value, because the [{value constraint}](#e-value_constraint) will override it.

[{identity-constraint definitions}](#identity-constraint_definitions) express constraints establishing uniquenesses and reference relationships among the values of related elements and attributes. See [Identity-constraint Definitions](#cIdentity-constraint_Definitions).

Element declarations are potential members of the substitution group, if any, identified by [{substitution group affiliation}](#class_exemplar). Potential membership is transitive but not symmetric; an element declaration is a potential member of any group of which its [{substitution group affiliation}](#class_exemplar) is a potential member. Actual membership may be blocked by the effects of [{substitution group exclusions}](#e-final) or [{disallowed substitutions}](#e-exact), see below.

An empty [{substitution group exclusions}](#e-final) allows a declaration to be nominated as the [{substitution group affiliation}](#class_exemplar) of other element declarations having the same [{type definition}](#type_definition) or types derived therefrom. The explicit values of [{substitution group exclusions}](#e-final) rule out element declarations having types which are extensions or restrictions respectively of [{type definition}](#type_definition). If both values are specified, then the declaration may not be nominated as the [{substitution group affiliation}](#class_exemplar) of any other declaration.

The supplied values for [{disallowed substitutions}](#e-exact) determine whether an element declaration appearing in a [content model](#key-contentModel) will be prevented from additionally [validating](#key-vn) elements (a) with an [xsi:type](#xsi_type) that identifies an extension or restriction of the type of the declared element, and/or (b) from [validating](#key-vn) elements which are in the substitution group headed by the declared element. If [{disallowed substitutions}](#e-exact) is empty, then all derived types and substitution group members are allowed.

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

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#e-annotation) property.

### XML Representation of Element Declaration Schema Components

The XML representation for an element declaration schema component is an [<element>](#element) 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:

**XML Representation Summary:** element **Element Information Item**

If the [<element>](#element) element information item has [<schema>](#schema) as its parent, the corresponding schema component is as follows:

[Element Declaration](#Element_Declaration_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element).attribute.

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent [<schema>](#schema) element information item, or [absent](#key-null) if there is none.

{scope} global.

{type definition} The type definition corresponding to the [<simpleType>](#simpleType) or [<complexType>](#complexType) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if either is present, otherwise the type definition children, if either is present, otherwise the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the type [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), otherwise the attribute, otherwise the [{type definition}](#type_definition) of the element declaration [resolved](#src-resolve) to by the [actual value](#key-vv) of the substitutionGroup [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise the attribute, if present, otherwise the [ur-type definition](#ur-type-itself).

{nillable} The [actual value](#key-vv) of the nillable [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise false.

{value constraint} If there is a default or a fixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), then a pair consisting of the attribute, then a pair consisting of the [actual value](#key-vv) (with respect to the [{type definition}](#type_definition), if it is a simple type definition, or the [{type definition}](#type_definition)'s [{content type}](#content_type), if that is a simple type definition, or else with respect to the built-in [string](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#string) simple type definition) of that [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) and either attribute and either default or fixed, as appropriate, otherwise [absent](#key-null).

{identity-constraint definitions} A set consisting of the identity-constraint-definitions corresponding to all the [<key>](#key), [<unique>](#unique) and [<keyref>](#keyref) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, otherwise the empty set.children, if any, otherwise the empty set.

{substitution group affiliation} The element declaration [resolved](#src-resolve) to by the [actual value](#key-vv) of the substitutionGroup [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise [absent](#key-null).

{disallowed substitutions} A set depending on the [actual value](#key-vv) of the block [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise on the attribute, if present, otherwise on the [actual value](#key-vv) of the blockDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the ancestor attribute of the ancestor [<schema>](#schema) 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

the EBV is the empty string

the empty set;

the EBV is #all

{extension, restriction, substitution};

a set with members drawn from the set above, each being present or absent depending on whether the [actual value](#key-vv) (which is a list) contains an equivalently named item.

**Note**

Although the blockDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of attribute of [<schema>](#schema) may include values other than extension, restriction or substitution, those values are ignored in the determination of [{disallowed substitutions}](#e-exact) for element declarations (they are used elsewhere).

{substitution group exclusions} As for [{disallowed substitutions}](#e-exact) above, but using the final and finalDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) in place of the attributes in place of the block and blockDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) and with the relevant set being attributes and with the relevant set being {extension, restriction}.

{abstract} The [actual value](#key-vv) of the abstract [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise false.

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

otherwise if the [<element>](#element) element information item has [<complexType>](#complexType) or [<group>](#group) as an ancestor and the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is absent, the corresponding schema components are as follows (unless 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):

[Particle](#Particle_details) **Schema Component**

**Property Representation**

{min occurs} The [actual value](#key-vv) of the minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{max occurs} unbounded, if the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) equals attribute equals unbounded, otherwise the [actual value](#key-vv) of the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{term} A (local) element declaration as given below.

An element declaration as in the first case above, with the exception of its [{target namespace}](#e-target_namespace) and [{scope}](#e-scope) properties, which are as below:

[Element Declaration](#Element_Declaration_details) **Schema Component**

**Property Representation**

{target namespace} If form is present and its [actual value](#key-vv) is qualified, or if form is absent and the [actual value](#key-vv) of elementFormDefault on the [<schema>](#schema) ancestor is qualified, then the [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent [<schema>](#schema) element information item, or [absent](#key-null) if there is none, otherwise [absent](#key-null).

{scope} If the [<element>](#element) element information item has [<complexType>](#complexType) as an ancestor, the complex definition corresponding to that item, otherwise (the [<element>](#element) element information item is within a named [<group>](#group) definition), [absent](#key-null).

otherwise (the [<element>](#element) element information item has [<complexType>](#complexType) or [<group>](#group) as an ancestor and the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present), the corresponding schema component is as follows (unless 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):

[Particle](#Particle_details) **Schema Component**

**Property Representation**

{min occurs} The [actual value](#key-vv) of the minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{max occurs} unbounded, if the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) equals attribute equals unbounded, otherwise the [actual value](#key-vv) of the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{term} The (top-level) element declaration [resolved](#src-resolve) to by the [actual value](#key-vv) of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element).attribute.

[<element>](#element) corresponds to an element declaration, and allows the type definition of that declaration to be specified either by reference or by explicit inclusion.

[<element>](#element)s within [<schema>](#schema) produce global element declarations; [<element>](#element)s within [<group>](#group) or [<complexType>](#complexType) 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 [<simpleType>](#simpleType) or [<complexType>](#complexType) is provided inline.

Element information items [validated](#key-vn) by a top-level declaration must be qualified with the [{target namespace}](#e-target_namespace) of that declaration (if this is [absent](#key-null), the item must be unqualified). Control over whether element information items [validated](#key-vn) by a local declaration must be similarly qualified or not is provided by the form [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), whose default is provided by the attribute, whose default is provided by the elementFormDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) on the enclosing attribute on the enclosing [<schema>](#schema), via its determination of [{target namespace}](#e-target_namespace).

As noted above the names for top-level element declarations are in a separate [symbol space](#key-symbolSpace) 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 [{target namespace}](#e-target_namespace) 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 [<element>](#element) 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](#key-urType). 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 [XML Representation of Identity-constraint Definition Schema Components](#declare-key) for [<key>](#key), [<unique>](#unique) and [<keyref>](#keyref).

**Example**

<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](#key-urType). 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.

**Note**

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.

**Example**

<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: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).

### Constraints on XML Representations of Element Declarations

### Element Declaration Validation Rules

**Note**

The [{name}](#e-name) and [{target namespace}](#e-target_namespace) properties are not mentioned above because they are checked during particle [validation](#key-vn), as per [Element Sequence Locally Valid (Particle)](#cvc-particle).

### Element Declaration Information Set Contributions

### Constraints on Element Declaration Schema Components

All element declarations (see [Element Declarations](#cElement_Declarations)) must satisfy the following constraint.

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

## Complex Type Definitions

Complex Type Definitions provide for:

Constraining element information items by providing [Attribute Declaration](#Attribute_Declaration)s governing the appearance and content of [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element)attributes

Constraining element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character 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](http://www.w3.org/TR/xml-infoset/#infoitem.element) to conform to a specified simple type definition.children to conform to a specified simple type definition.

Using the mechanisms of [Type Definition Hierarchy](#Type_Derivation) to derive a complex type from another simple or complex type.

Specifying [post-schema-validation infoset contributions](#gloss-sic) 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.

**Example**

<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.

### The Complex Type Definition Schema Component

A complex type definition schema component has the following properties:

**Schema Component:** [Complex Type Definition](#Complex_Type_Definition)

{name} Optional. An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{base type definition} Either a simple type definition or a complex type definition.

{derivation method} Either extension or restriction.

{final} A subset of {extension, restriction}.

{abstract} A boolean

{attribute uses} A set of attribute uses.

{attribute wildcard} Optional. A wildcard.

{content type} One of empty, a simple type definition or a pair consisting of a [content model](#key-contentModel) (I.e. a [Particle](#Particle)) and one of mixed, element-only.

{prohibited substitutions} A subset of {extension, restriction}.

{annotations} A set of annotations.

Complex types definitions are identified by their [{name}](#ct-name) and [{target namespace}](#ct-target_namespace). Except for anonymous complex type definitions (those with no [{name}](#ct-name)), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an [XML Schema](#key-schema), no complex type definition can have the same name as another simple or complex type definition. Complex type [{name}](#ct-name)s and [{target namespace}](#ct-target_namespace)s are provided for reference from instances (see [xsi:type](#xsi_type)), and for use in the XML representation of schema components (specifically in [<element>](#element)). See [References to schema components across namespaces](#composition-schemaImport) for the use of component identifiers when importing one schema into another.

**Note**

The [{name}](#ct-name) of a complex type is not ipso facto the [(local) name](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the element information items (local) name of the element information items [validated](#key-vn) by that definition. The connection between a name and a type definition is described in [Element Declarations](#cElement_Declarations).

As described in [Type Definition Hierarchy](#Type_Derivation), each complex type is derived from a [{base type definition}](#ct-base_type_definition) which is itself either a [Simple Type Definition](#Simple_Type_Definition) or a [Complex Type Definition](#Complex_Type_Definition). [{derivation method}](#derivation_method) specifies the means of derivation as either extension or restriction (see [Type Definition Hierarchy](#Type_Derivation)).

A complex type with an empty specification for [{final}](#ct-final) can be used as a [{base type definition}](#ct-base_type_definition) 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 [Definition:] 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 [{abstract}](#ct-abstract) is true must not be used as the [{type definition}](#type_definition) for the [validation](#key-vn) of element information items. It follows that they must not be referenced from an [xsi:type](#xsi_type) attribute in an instance document. Abstract complex types can be used as [{base type definition}](#ct-base_type_definition)s, or even as the [{type definition}](#type_definition)s of element declarations, provided in every case a concrete derived type definition is used for [validation](#key-vn), either via [xsi:type](#xsi_type) or the operation of a substitution group.

[{attribute uses}](#ct-attribute_declarations) are a set of attribute uses. See [Element Locally Valid (Complex Type)](#cvc-complex-type) and [Attribute Locally Valid](#cvc-attribute) for details of attribute [validation](#key-vn).

[{attribute wildcard}](#ct-attribute_wildcard)s provide a more flexible specification for [validation](#key-vn) of attributes not explicitly included in [{attribute uses}](#ct-attribute_declarations). Informally, the specific values of [{attribute wildcard}](#ct-attribute_wildcard) are interpreted as follows: See [Element Locally Valid (Complex Type)](#cvc-complex-type) and [Wildcard allows Namespace Name](#cvc-wildcard-namespace) for formal details of attribute wildcard [validation](#key-vn).

any: [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) can include attributes with any qualified or unqualified name.attributes can include attributes with any qualified or unqualified name.

a set whose members are either namespace names or [absent](#key-null): [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) can include any attribute(s) from the specified namespace(s). If attributes can include any attribute(s) from the specified namespace(s). If [absent](#key-null) is included in the set, then any unqualified attributes are (also) allowed.

'not' and a namespace name: [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) cannot include attributes from the specified namespace.attributes cannot include attributes from the specified namespace.

'not' and [absent](#key-null): [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) cannot include unqualified attributes.attributes cannot include unqualified attributes.

[{content type}](#content_type) determines the [validation](#key-vn) of [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of element information items. Informally: children of element information items. Informally:

A [{content type}](#content_type) with the distinguished value empty [validates](#key-vn) elements with no character or element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element).children.

A [{content type}](#content_type) which is a [Simple Type Definition](#Simple_Type_Definition) [validates](#key-vn) elements with character-only [children](http://www.w3.org/TR/xml-infoset/#infoitem.element).children.

An element-only [{content type}](#content_type) [validates](#key-vn) elements with [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) that conform to the supplied children that conform to the supplied [content model](#key-contentModel).

A mixed [{content type}](#content_type) [validates](#key-vn) elements whose element [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) (i.e. specifically ignoring other children (i.e. specifically ignoring other [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) such as character information items) conform to the supplied children such as character information items) conform to the supplied [content model](#key-contentModel).

[{prohibited substitutions}](#ct-exact) determine whether an element declaration appearing in a [content model](#key-contentModel) is prevented from additionally [validating](#key-vn) element items with an [xsi:type](#xsi_type) 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 [{prohibited substitutions}](#ct-exact) is empty, then all such substitutions are allowed, otherwise, the derivation method(s) it names are disallowed.

See [Annotations](#cAnnotations) for information on the role of the [{annotations}](#ct-annotations) property.

### XML Representation of Complex Type Definitions

The XML representation for a complex type definition schema component is a [<complexType>](#complexType) element information item.

The XML representation for complex type definitions with a simple type definition [{content type}](#content_type) is significantly different from that of those with other [{content type}](#content_type)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.

**XML Representation Summary:** complexType **Element Information Item**

Whichever alternative for the content of [<complexType>](#complexType) is chosen, the following property mappings apply:

[Complex Type Definition](#Complex_Type_Definition_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) if present, otherwise attribute if present, otherwise [absent](#key-null).

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<schema>](#schema) ancestor element information item if present, otherwise [absent](#key-null).

{abstract} The [actual value](#key-vv) of the abstract [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise false.

{prohibited substitutions} A set corresponding to the [actual value](#key-vv) of the block [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise on the attribute, if present, otherwise on the [actual value](#key-vv) of the blockDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the ancestor attribute of the ancestor [<schema>](#schema) 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

the EBV is the empty string

the empty set;

the EBV is #all

{extension, restriction};

a set with members drawn from the set above, each being present or absent depending on whether the [actual value](#key-vv) (which is a list) contains an equivalently named item.

**Note**

Although the blockDefault [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of attribute of [<schema>](#schema) may include values other than restriction orextension, those values are ignored in the determination of [{prohibited substitutions}](#ct-exact) for complex type definitions (they are used elsewhere).

{final} As for [{prohibited substitutions}](#ct-exact) above, but using the final and finalDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) in place of the attributes in place of the block and blockDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element).attributes.

{annotations} The annotations corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, in the children, if present, in the [<simpleContent>](#simpleContent) and [<complexContent>](#complexContent) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, and in their children, if present, and in their [<restriction>](#restriction) and [<extension>](#extension) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

When the [<simpleContent>](#simpleContent) alternative is chosen, the following elements are relevant, and the remaining property mappings are as below. Note that either [<restriction>](#restriction) or [<extension>](#extension) must be chosen as the content of [<simpleContent>](#simpleContent).

**XML Representation Summary:** simpleContent **Element Information Item**

**XML Representation Summary:** restriction **Element Information Item**

**XML Representation Summary:** extension **Element Information Item**

**XML Representation Summary:** attributeGroup **Element Information Item**

**XML Representation Summary:** anyAttribute **Element Information Item**

[Complex Type Definition with simple content](#Complex_Type_Definition_details) **Schema Component**

**Property Representation**

{base type definition} The type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{derivation method} If the [<restriction>](#restriction) alternative is chosen, then restriction, otherwise (the [<extension>](#extension) alternative is chosen) extension.

{attribute uses} A union of sets of attribute uses as follows

The set of attribute uses corresponding to the [<attribute>](#attribute) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

The [{attribute uses}](#ag-attribute_declarations) of the attribute groups [resolved](#src-resolve) to by the [actual value](#key-vv)s of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

if the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a complex type definition, the attribute is a complex type definition, the [{attribute uses}](#ct-attribute_declarations) of that type definition, unless the [<restriction>](#restriction) alternative is chosen, in which case some members of that type definition's [{attribute uses}](#ct-attribute_declarations) may not be included, namely those whose [{attribute declaration}](#attribute)'s [{name}](#a-name) and [{target namespace}](#a-target_namespace) are the same as

the [{name}](#a-name) and [{target namespace}](#a-target_namespace) of the [{attribute declaration}](#attribute) of an attribute use in the set per

what would have been the [{name}](#a-name) and [{target namespace}](#a-target_namespace) of the [{attribute declaration}](#attribute) of an attribute use in the set per [actual value](#key-vv) of the use [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the relevant attribute of the relevant [<attribute>](#attribute) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) being prohibited.

{attribute wildcard}

[Definition:] Let the local wildcard be defined as

there is an [<anyAttribute>](#anyAttribute) present

a wildcard based on the [actual value](#key-vv)s of the namespace and processContents [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) and the attributes and the [<annotation>](#annotation) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), exactly as for the wildcard corresponding to an children, exactly as for the wildcard corresponding to an [<any>](#any) element as set out in [XML Representation of Wildcard Schema Components](#declare-openness);

[absent](#key-null).

[Definition:] Let the complete wildcard be defined as

there are no [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) corresponding to attribute groups with non-children corresponding to attribute groups with non-[absent](#key-null) [{attribute wildcard}](#ag-attribute_wildcard)s

the [local wildcard](#key-law).

there are one or more [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) corresponding to attribute groups with non-children corresponding to attribute groups with non-[absent](#key-null) [{attribute wildcard}](#ag-attribute_wildcard)s

there is an [<anyAttribute>](#anyAttribute) present

a wildcard whose [{process contents}](#process_contents) and [{annotation}](#w-annotation) are those of the [local wildcard](#key-law), and whose [{namespace constraint}](#namespace_constraint) is the intensional intersection of the [{namespace constraint}](#namespace_constraint) of the [local wildcard](#key-law) and of the [{namespace constraint}](#namespace_constraint)s of all the non-[absent](#key-null) [{attribute wildcard}](#ag-attribute_wildcard)s of the attribute groups corresponding to the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), as defined in children, as defined in [Attribute Wildcard Intersection](#cos-aw-intersect).

there is no [<anyAttribute>](#anyAttribute) present

a wildcard whose properties are as follows:

[{process contents}](#process_contents)

The [{process contents}](#process_contents) of the first non-[absent](#key-null) [{attribute wildcard}](#ag-attribute_wildcard) of an attribute group among the attribute groups corresponding to the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element).children.

[{namespace constraint}](#namespace_constraint)

The intensional intersection of the [{namespace constraint}](#namespace_constraint)s of all the non-[absent](#key-null) [{attribute wildcard}](#ag-attribute_wildcard)s of the attribute groups corresponding to the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), as defined in children, as defined in [Attribute Wildcard Intersection](#cos-aw-intersect).

[{annotation}](#w-annotation)

[absent](#key-null).

The value is then determined by

the [<restriction>](#restriction) alternative is chosen

the [complete wildcard](#key-eaw);

the [<extension>](#extension) alternative is chosen

[Definition:] let the base wildcard be defined as

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a complex type definition with an attribute is a complex type definition with an [{attribute wildcard}](#ct-attribute_wildcard)

that [{attribute wildcard}](#ct-attribute_wildcard).

[absent](#key-null).

The value is then determined by

the [base wildcard](#key-baw) is non-[absent](#key-null)

the [complete wildcard](#key-eaw) is [absent](#key-null)

the [base wildcard](#key-baw).

a wildcard whose [{process contents}](#process_contents) and [{annotation}](#w-annotation) are those of the [complete wildcard](#key-eaw), and whose [{namespace constraint}](#namespace_constraint) is the intensional union of the [{namespace constraint}](#namespace_constraint) of the [complete wildcard](#key-eaw) and of the [base wildcard](#key-baw), as defined in [Attribute Wildcard Union](#cos-aw-union).

(the [base wildcard](#key-baw) is [absent](#key-null)) the [complete wildcard](#key-eaw)

{content type}

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a complex type definition whose own attribute is a complex type definition whose own [{content type}](#content_type) is a simple type definition and the [<restriction>](#restriction) alternative is chosen

starting from either [<restriction>](#restriction)'s [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) (i.e. those which specify facets, if any), as defined in children (i.e. those which specify facets, if any), as defined in [Simple Type Restriction (Facets)](#st-restrict-facets);

the simple type definition corresponding to the [<simpleType>](#simpleType) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) if there is one;

otherwise ([<restriction>](#restriction) has no [<simpleType>](#simpleType) among its [children](http://www.w3.org/TR/xml-infoset/#infoitem.element)), the simple type definition which is the children), the simple type definition which is the [{content type}](#content_type) of the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a complex type definition whose own attribute is a complex type definition whose own [{content type}](#content_type) is mixed and a particle which is [emptiable](#cd-emptiable), as defined in [Particle Emptiable](#cos-group-emptiable) and the [<restriction>](#restriction) alternative is chosen

starting from the simple type definition corresponding to the [<simpleType>](#simpleType) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) (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 [<restriction>](#restriction)'s [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) (i.e. those which specify facets, if any), as defined in children (i.e. those which specify facets, if any), as defined in [Simple Type Restriction (Facets)](#st-restrict-facets);

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a complex type definition (whose own attribute is a complex type definition (whose own [{content type}](#content_type) must be a simple type definition, see below) and the [<extension>](#extension) alternative is chosen

the [{content type}](#content_type) of that complex type definition;

(the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is a simple type definition and the attribute is a simple type definition and the [<extension>](#extension) alternative is chosen), then that simple type definition.

When the [<complexContent>](#complexContent) alternative is chosen, the following elements are relevant (as are the [<attributeGroup>](#attributeGroup) and [<anyAttribute>](#anyAttribute) elements, not repeated here), and the additional property mappings are as below. Note that either [<restriction>](#restriction) or [<extension>](#extension) must be chosen as the content of [<complexContent>](#complexContent), but their content models are different in this case from the case above when they occur as children of [<simpleContent>](#simpleContent).

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

**XML Representation Summary:** complexContent **Element Information Item**

**XML Representation Summary:** restriction **Element Information Item**

**XML Representation Summary:** extension **Element Information Item**

[Complex Type Definition with complex content](#Complex_Type_Definition_details) **Schema Component**

**Property Representation**

{base type definition} The type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{derivation method} If the [<restriction>](#restriction) alternative is chosen, then restriction, otherwise (the [<extension>](#extension) alternative is chosen) extension.

{attribute uses} A union of sets of attribute uses as follows:

The set of attribute uses corresponding to the [<attribute>](#attribute) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

The [{attribute uses}](#ag-attribute_declarations) of the attribute groups [resolved](#src-resolve) to by the [actual value](#key-vv)s of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

The [{attribute uses}](#ct-attribute_declarations) of the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), unless the attribute, unless the [<restriction>](#restriction) alternative is chosen, in which case some members of that type definition's [{attribute uses}](#ct-attribute_declarations) may not be included, namely those whose [{attribute declaration}](#attribute)'s [{name}](#a-name) and [{target namespace}](#a-target_namespace) are the same as

The [{name}](#a-name) and [{target namespace}](#a-target_namespace) of the [{attribute declaration}](#attribute) of an attribute use in the set per

what would have been the [{name}](#a-name) and [{target namespace}](#a-target_namespace) of the [{attribute declaration}](#attribute) of an attribute use in the set per [actual value](#key-vv) of the use [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the relevant attribute of the relevant [<attribute>](#attribute) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) being prohibited.

{attribute wildcard} As above for the [<simpleContent>](#simpleContent) alternative.

{content type}

[Definition:] Let the effective mixed be

the mixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present on attribute is present on [<complexContent>](#complexContent)

its [actual value](#key-vv);

the mixed [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present on attribute is present on [<complexType>](#complexType)

its [actual value](#key-vv);

elementOnlyfalse.

[Definition:] Let the effective content be

There is no [<group>](#group), [<all>](#all), [<choice>](#choice) or [<sequence>](#sequence) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element);children;

There is an [<all>](#all) or [<sequence>](#sequence) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) with no children with no [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of its own excluding children of its own excluding [<annotation>](#annotation);

There is a [<choice>](#choice) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) with no children with no [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of its own excluding children of its own excluding [<annotation>](#annotation) whose minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) has the attribute has the [actual value](#key-vv) 0;

empty

the [effective mixed](#key-efm) is true

A particle whose properties are as follows:

[{min occurs}](#p-min_occurs)

1

[{max occurs}](#p-max_occurs)

1

[{term}](#term)

A model group whose [{compositor}](#compositor) is sequence and whose [{particles}](#particles) is empty.

empty

the particle corresponding to the [<all>](#all), [<choice>](#choice), [<group>](#group) or [<sequence>](#sequence) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element).children.

Then the value of the property is

the [<restriction>](#restriction) alternative is chosen

the [effective content](#key-exg) is empty

empty;

a pair consisting of

mixed if the [effective mixed](#key-efm) is true, otherwise elementOnly

The [effective content](#key-exg).

the [<extension>](#extension) alternative is chosen

the [effective content](#key-exg) is empty

the [{content type}](#content_type) of the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) has a attribute has a [{content type}](#content_type) of empty

a pair as per

a pair of mixed or elementOnly (determined as per

[{min occurs}](#p-min_occurs)

1

[{max occurs}](#p-max_occurs)

1

[{term}](#term)

A model group whose [{compositor}](#compositor) is sequence and whose [{particles}](#particles) are the particle of the [{content type}](#content_type) of the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) followed by the attribute followed by the [explicit contenteffective content](#key-exg).

**Note**

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

**Note**

The only substantive function of the value prohibited for the use attribute of an [<attribute>](#attribute) 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 [{base type definition}](#ct-base_type_definition). Such an [<attribute>](#attribute) does not correspond to any component, and hence there is no interaction with either explicit or inherited wildcards in the operation of [Complex Type Definition Validation Rules](#formal-complex-type) or [Constraints on Complex Type Definition Schema Components](#coss-ct).

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.

**Example**

<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.

**Example**

<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](#key-vn) instance thereof.

**Example**

<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](#key-vn) instance thereof.

**Example**

<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.

### Constraints on XML Representations of Complex Type Definitions

### Complex Type Definition Validation Rules

### Complex Type Definition Information Set Contributions

### Constraints on Complex Type Definition Schema Components

All complex type definitions (see [Complex Type Definitions](#Complex_Type_Definitions)) must satisfy the following constraints.

**Note**

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.

**Note**

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.

**Note**

The wording of

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.

### Built-in Complex Type Definition

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

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

**Note**

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>.

## 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.

**Example**

<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.

### The Attribute Use Schema Component

The attribute use schema component has the following properties:

**Schema Component:** [Attribute Use](#Attribute_Use)

{required} A boolean.

{attribute declaration} An attribute declaration.

{value constraint} Optional. A pair consisting of a value and one of default, fixed.

[{required}](#required) determines whether this use of an attribute declaration requires an appropriate attribute information item to be present, or merely allows it.

[{attribute declaration}](#attribute) provides the attribute declaration itself, which will in turn determine the simple type definition used.

[{value constraint}](#au-value_constraint) allows for local specification of a default or fixed value. This must be consistent with that of the [{attribute declaration}](#attribute), in that if the [{attribute declaration}](#attribute) specifies a fixed value, the only allowed [{value constraint}](#au-value_constraint) is the same fixed value.

### XML Representation of Attribute Use Components

Attribute uses correspond to all uses of [<attribute>](#attribute) which allow a use attribute. These in turn correspond to two components in each case, an attribute use and its [{attribute declaration}](#attribute) (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 [XML Representation of Attribute Declaration Schema Components](#declare-attribute).

### Constraints on XML Representations of Attribute Uses

None as such.

### Attribute Use Validation Rules

### Attribute Use Information Set Contributions

None as such.

### Constraints on Attribute Use Schema Components

All attribute uses (see [AttributeUses](#cAttributeUse)) must satisfy the following constraints.

## 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](#key-vn) as such, but the [{attribute uses}](#ct-attribute_declarations) and [{attribute wildcard}](#ct-attribute_wildcard) 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](http://www.w3.org/TR/REC-xml#dt-PE) facility. Attribute group definitions are provided primarily for reference from the XML representation of schema components (see [<complexType>](#complexType) and [<attributeGroup>](#attributeGroup)).

**Example**

<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.

### The Attribute Group Definition Schema Component

The attribute group definition schema component has the following properties:

**Schema Component:** [Attribute Group Definition](#Attribute_Group_Definition)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{attribute uses} A set of attribute uses.

{attribute wildcard} Optional. A wildcard.

{annotation} Optional. An annotation.

Attribute groups are identified by their [{name}](#ag-name) and [{target namespace}](#ag-target_namespace); attribute group identities must be unique within an [XML Schema](#key-schema). See [References to schema components across namespaces](#composition-schemaImport) for the use of component identifiers when importing one schema into another.

[{attribute uses}](#ag-attribute_declarations) is a set attribute uses, allowing for local specification of occurrence and default or fixed values.

[{attribute wildcard}](#ag-attribute_wildcard) provides for an attribute wildcard to be included in an attribute group. See above under [Complex Type Definitions](#Complex_Type_Definitions) for the interpretation of attribute wildcards during [validation](#key-vn).

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#ag-annotation) property.

### XML Representation of Attribute Group Definition Schema Components

The XML representation for an attribute group definition schema component is an [<attributeGroup>](#attributeGroup) 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:

**XML Representation Summary:** attributeGroup **Element Information Item**

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

[Attribute Group Definition](#Attribute_Group_Definition_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent schema element information item.

{attribute uses} The union of the set of attribute uses corresponding to the [<attribute>](#attribute) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, with the children, if any, with the [{attribute uses}](#ag-attribute_declarations) of the attribute groups [resolved](#src-resolve) to by the [actual value](#key-vv)s of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<attributeGroup>](#attributeGroup) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

{attribute wildcard} As for the [complete wildcard](#key-eaw) as described in [XML Representation of Complex Type Definitions](#declare-type).

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

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.

### Constraints on XML Representations of Attribute Group Definitions

### Attribute Group Definition Validation Rules

None as such.

### Attribute Group Definition Information Set Contributions

None as such.

### Constraints on Attribute Group Definition Schema Components

All attribute group definitions (see [Attribute Group Definitions](#cAttribute_Group_Definitions)) must satisfy the following constraint.

## Model Group Definitions

A model group definition associates a name and optional annotations with a [Model Group](#Model_Group). By reference to the name, the entire model group can be incorporated by reference into a [{term}](#term).

Model group definitions are provided primarily for reference from the [XML Representation of Complex Type Definitions](#declare-type) (see [<complexType>](#complexType) and [<group>](#group)). Thus, model group definitions provide a replacement for some uses of XML's [parameter entity](http://www.w3.org/TR/REC-xml#dt-PE) facility.

**Example**

<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.

### The Model Group Definition Schema Component

The model group definition schema component has the following properties:

**Schema Component:** [Model Group Definition](#Model_Group_Definition)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{model group} A model group.

{annotation} Optional. An annotation.

Model group definitions are identified by their [{name}](#mg-name) and [{target namespace}](#mg-target_namespace); model group identities must be unique within an [XML Schema](#key-schema). See [References to schema components across namespaces](#composition-schemaImport) for the use of component identifiers when importing one schema into another.

Model group definitions per se do not participate in [validation](#key-vn), but the [{term}](#term) of a particle may correspond in whole or in part to a model group from a model group definition.

[{model group}](#model_group) is the [Model Group](#Model_Group) for which the model group definition provides a name.

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#mg-annotation) property.

### XML Representation of Model Group Definition Schema Components

The XML representation for a model group definition schema component is a [<group>](#group) 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:

**XML Representation Summary:** group **Element Information Item**

If there is a name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) (in which case the item will have attribute (in which case the item will have [<schema>](#schema) or [<redefine>](#redefine) as parent), then the item corresponds to a model group definition component with properties as follows:

[Model Group Definition](#Model_Group_Definition_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent schema element information item.

{model group} A model group which is the [{term}](#term) of a particle corresponding to the [<all>](#all), [<choice>](#choice) or [<sequence>](#sequence) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) (there must be one).children (there must be one).

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

Otherwise, the item will have a ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), in which case it corresponds to a particle component with properties as follows (unless 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):

[Particle](#Particle) **Schema Component**

**Property Representation**

{min occurs} The [actual value](#key-vv) of the minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{max occurs} unbounded, if the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) equals attribute equals unbounded, otherwise the [actual value](#key-vv) of the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{term} The [{model group}](#model_group) of the model group definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the ref [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)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 [<group>](#group) when it is named. This in turn is because the [{min occurs}](#p-min_occurs) and [{max occurs}](#p-max_occurs) of the particles which refer to the definition are what count.

Given the constraints on its appearance in content models, an [<all>](#all) should only occur as the only item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of a named model group definition or a content model: see children of a named model group definition or a content model: see [Constraints on Model Group Schema Components](#coss-modelGroup).

### Constraints on XML Representations of Model Group Definitions

### Model Group Definition Validation Rules

None as such.

### Model Group Definition Information Set Contributions

None as such.

### Constraints on Model Group Definition Schema Components

All model group definitions (see [Model Group Definitions](#cModel_Group_Definitions)) must satisfy the following constraint.

## Model Groups

When the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of element information items are not constrained to be children of element information items are not constrained to be empty or by reference to a simple type definition ([Simple Type Definitions](#Simple_Type_Definitions)), the sequence of element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) content may be specified in more detail with a model group. Because the children content may be specified in more detail with a model group. Because the [{term}](#term) 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.

**Example**

<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.

### The Model Group Schema Component

The model group schema component has the following properties:

**Schema Component:** [Model Group](#Model_Group)

{compositor} One of all, choice or sequence.

{particles} A list of particles

{annotation} Optional. An annotation.

specifies a sequential (sequence), disjunctive (choice) or conjunctive (all) interpretation of the [{particles}](#particles). This in turn determines whether the element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) children [validated](#key-vn) by the model group must:

(sequence) correspond, in order, to the specified [{particles}](#particles);

(choice) corresponded to exactly one of the specified [{particles}](#particles);

(all) contain all and only exactly zero or one of each element specified in [{particles}](#particles). The elements can occur in any order. In this case, to reduce implementation complexity, [{particles}](#particles) is restricted to contain local and top-level element declarations only, with [{min occurs}](#p-min_occurs)=0 or 1, [{max occurs}](#p-max_occurs)=1.

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

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#amg-annotation) property.

### XML Representation of Model Group Schema Components

The XML representation for a model group schema component is either an [<all>](#all), a [<choice>](#choice) or a [<sequence>](#sequence) element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

**XML Representation Summary:** all **Element Information Item**

**XML Representation Summary:** choice **Element Information Item**

**XML Representation Summary:** sequence **Element Information Item**

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):

[Particle](#Particle_details) **Schema Component**

**Property Representation**

{min occurs} The [actual value](#key-vv) of the minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{max occurs} unbounded, if the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) equals attribute equals unbounded, otherwise the [actual value](#key-vv) of the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{term} A model group as given below:

[Model Group](#Model_Group_details) **Schema Component**

**Property Representation**

{compositor} One of all, choice, sequence depending on the element information item.

{particles} A sequence of particles corresponding to all the [<all>](#all), [<choice>](#choice), [<sequence>](#sequence), [<any>](#any), [<group>](#group) or [<element>](#element) items among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), in order.children, in order.

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

### Constraints on XML Representations of Model Groups

### Model Group Validation Rules

**Note**

The above definition is implicitly non-deterministic, and should not be taken as a recipé for implementations. Note in particular that when [{compositor}](#compositor) is all, particles is restricted to a list of local and top-level element declarations (see [Constraints on Model Group Schema Components](#coss-modelGroup)). A much simpler implementation is possible than would arise from a literal interpretation of the definition above; informally, the content is [valid](#key-vn) when each declared element occurs exactly once (or at most once, if [{min occurs}](#p-min_occurs) is 0), and each is [valid](#key-vn) with respect to its corresponding declaration. The elements can occur in arbitrary order.

### Model Group Information Set Contributions

None as such.

### Constraints on Model Group Schema Components

All model groups (see [Model Groups](#Model_Groups)) must satisfy the following constraints.

**Note**

Because locally-scoped element declarations may or may not have a [{target namespace}](#e-target_namespace), 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.

## Particles

As described in [Model Groups](#Model_Groups), particles contribute to the definition of content models.

**Example**

<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.

### The Particle Schema Component

The particle schema component has the following properties:

**Schema Component:** [Particle](#Particle)

{min occurs} A non-negative integer.

{max occurs} Either a non-negative integer or unbounded.

{term} One of a model group, a wildcard, or an element declaration.

In general, multiple element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), possibly with intervening character children, possibly with intervening character [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) if the content type is children if the content type is mixed, can be [validated](#key-vn) with respect to a single particle. When the [{term}](#term) is an element declaration or wildcard, [{min occurs}](#p-min_occurs) determines the minimum number of such element [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) that can occur. The number of such children must be greater than or equal to children that can occur. The number of such children must be greater than or equal to [{min occurs}](#p-min_occurs). If [{min occurs}](#p-min_occurs) is 0, then occurrence of such children is optional.

Again, when the [{term}](#term) is an element declaration or wildcard, the number of such element [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) must be less than or equal to any numeric specification of children must be less than or equal to any numeric specification of [{max occurs}](#p-max_occurs); if [{max occurs}](#p-max_occurs) is unbounded, then there is no upper bound on the number of such children.

When the [{term}](#term) is a model group, the permitted occurrence range is determined by a combination of [{min occurs}](#p-min_occurs) and [{max occurs}](#p-max_occurs) and the occurrence ranges of the [{term}](#term)'s [{particles}](#particles).

### XML Representation of Particle Components

Particles correspond to all three elements ([<element>](#element) not immediately within [<schema>](#schema), [<group>](#group) not immediately within [<schema>](#schema) and [<any>](#any)) which allow minOccurs and maxOccurs attributes. These in turn correspond to two components in each case, a particle and its [{term}](#term). The appropriate mapping is described in [XML Representation of Element Declaration Schema Components](#declare-element), [XML Representation of Model Group Schema Components](#declare-contentModel) and [XML Representation of Wildcard Schema Components](#declare-openness) respectively.

### Constraints on XML Representations of Particles

None as such.

### Particle Validation Rules

### Particle Information Set Contributions

None as such.

### Constraints on Particle Schema Components

All particles (see [Particles](#cParticles)) must satisfy the following constraints.

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

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.

**Pf**

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.

## 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](#key-vn) of attribute and element information items dependent on their namespace name, but independently of their local name.

**Example**

<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.

### The Wildcard Schema Component

The wildcard schema component has the following properties:

**Schema Component:** [Wildcard](#Wildcard)

{namespace constraint} One of any; a pair of not and a namespace name or [absent](#key-null); or a set whose members are either namespace names or [absent](#key-null).

{process contents} One of skip, lax or strict.

{annotation} Optional. An annotation.

[{namespace constraint}](#namespace_constraint) provides for [validation](#key-vn) of attribute and element items that:

(any) have any namespace or are not namespace-qualified;

(not and a namespace name) are namespace-qualified with a namespace other than the specified namespace name;

(not and [absent](#key-null)) are namespace-qualified;

(a set whose members are either namespace names or [absent](#key-null)) have any of the specified namespaces and/or, if [absent](#key-null) is included in the set, are unqualified.

[{process contents}](#process_contents) controls the impact on [assessment](#key-va) 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](#key-vn) 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](#key-vn) with respect to that definition, that is, [validate](#key-vn) if you can, don't worry if you can't.

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#w-annotation) property.

### XML Representation of Wildcard Schema Components

The XML representation for a wildcard schema component is an [<any>](#any) or [<anyAttribute>](#anyAttribute) element information item. The correspondences between the properties of an [<any>](#any) information item and properties of the components it corresponds to are as follows (see [<complexType>](#complexType) and [<attributeGroup>](#attributeGroup) for the correspondences for [<anyAttribute>](#anyAttribute)):

**XML Representation Summary:** any **Element Information Item**

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

[Particle](#Particle_details) **Schema Component**

**Property Representation**

{min occurs} The [actual value](#key-vv) of the minOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{max occurs} unbounded, if the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) equals attribute equals unbounded, otherwise the [actual value](#key-vv) of the maxOccurs [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise 1.

{term} A wildcard as given below:

[Wildcard](#Wildcard_details) **Schema Component**

**Property Representation**

{namespace constraint} Dependent on the [actual value](#key-vv) of the namespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element): if absent, then attribute: if absent, then any, otherwise as follows:

**##any**

any

**##other**

a pair of not and the [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<schema>](#schema) ancestor element information item if present, otherwise [absent](#key-null).

otherwise

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

if one such substring is ##targetNamespace, the corresponding member is the [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<schema>](#schema) ancestor element information item if present, otherwise [absent](#key-null).

if one such substring is ##local, the corresponding member is [absent](#key-null).

{process contents} The [actual value](#key-vv) of the processContents [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise strict.

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

Wildcards are subject to the same ambiguity constraints ([Unique Particle Attribution](#cos-nonambig)) 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.

### Constraints on XML Representations of Wildcards

### Wildcard Validation Rules

### Wildcard Information Set Contributions

None as such.

### Constraints on Wildcard Schema Components

All wildcards (see [Wildcards](#Wildcards)) must satisfy the following constraint.

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

## Identity-constraint Definitions

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

**Example**

<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.

### The Identity-constraint Definition Schema Component

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

**Schema Component:** [Identity-constraint Definition](#Identity-constraint_Definition)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{identity-constraint category} One of key, keyref or unique.

{selector} A restricted XPath ([[XPath]](#bib-xpath)) expression.

{fields} A non-empty list of restricted XPath ([[XPath]](#bib-xpath)) expressions.

{referenced key} Required if [{identity-constraint category}](#identity-constraint_name) is keyref, forbidden otherwise. An identity-constraint definition with [{identity-constraint category}](#identity-constraint_name) equal to key or unique.

{annotation} Optional. A set of annotations.

Identity-constraint definitions are identified by their [{name}](#c-name) and [{target namespace}](#c-target_namespace); Identity-constraint definition identities must be unique within an [XML Schema](#key-schema). See [References to schema components across namespaces](#composition-schemaImport) for the use of component identifiers when importing one schema into another.

Informally, [{identity-constraint category}](#identity-constraint_name) 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 [{selector}](#selector), of the tuples resulting from evaluation of the [{fields}](#fields) 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 [{selector}](#selector), of the tuples resulting from evaluation of the [{fields}](#fields) XPath expression(s), with those of the [{referenced key}](#referenced_key).

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-nillable;

The comparison between keyref [{fields}](#fields) and key or unique [{fields}](#fields) is by value equality, not by string equality.

[{selector}](#selector) specifies a restricted XPath ([[XPath]](#bib-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.

[{fields}](#fields) specifies XPath expressions relative to each element selected by a [{selector}](#selector). 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 [{fields}](#fields)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 [{selector}](#selector) and [{fields}](#fields). The details are given in [Constraints on Identity-constraint Definition Schema Components](#coss-identity-constraint).

**Note**

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

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#rc-annotation) property.

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

The XML representation for an identity-constraint definition schema component is either a [<key>](#key), a [<keyref>](#keyref) or a [<unique>](#unique) element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

**XML Representation Summary:** unique **Element Information Item**

**XML Representation Summary:** key **Element Information Item**

**XML Representation Summary:** keyref **Element Information Item**

**XML Representation Summary:** selector **Element Information Item**

**XML Representation Summary:** field **Element Information Item**

[Identity-constraint Definition](#Identity-constraint_Definition_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent schema element information item.

{identity-constraint category} One of key, keyref or unique, depending on the item.

{selector} A restricted XPath expression corresponding to the [actual value](#key-vv) of the xpath [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<selector>](#selector) element information item among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element)children

{fields} A sequence of XPath expressions, corresponding to the [actual value](#key-vv)s of the xpath [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)s of the attributes of the [<field>](#field) element information item [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), in order.children, in order.

{referenced key} If the item is a [<keyref>](#keyref), the identity-constraint definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the refer [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), otherwise attribute, otherwise [absent](#key-null).

{annotation} The annotations corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, and in the children, if present, and in the [<selector>](#selector) and [<field>](#field) [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

**Example**

<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.

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

### Identity-constraint Definition Validation Rules

**Note**

Because the validation of keyref (see [node table](#key-nt), and [node tables](#key-nt) 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](#key-vn) can be referenced successfully.

**Note**

Although this specification defines a [post-schema-validation infoset](#key-psvi) contribution which would enable schema-aware processors to implement [Element Declaration](#sic-elt-decl)), 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 [{nillable}](#nillable) property must be available.

### Identity-constraint Definition Information Set Contributions

### Constraints on Identity-constraint Definition Schema Components

All identity-constraint definitions (see [Identity-constraint Definitions](#cIdentity-constraint_Definitions)) must satisfy the following constraint.

## Notation Declarations

Notation declarations reconstruct XML 1.0 NOTATION declarations.

**Example**

<xs:notation name="jpeg" public="image/jpeg" system="viewer.exe">

The XML representation of a notation declaration.

### The Notation Declaration Schema Component

The notation declaration schema component has the following properties:

**Schema Component:** [Notation Declaration](#Notation_Declaration)

{name} An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{system identifier} Optional if [{public identifier}](#public_identifier) is present. A URI reference.

{public identifier} Optional if [{system identifier}](#system_identifier) is present. A public identifier, as defined in [[XML 1.0 (Second Edition)]](#ref-xml).

{annotation} Optional. An annotation.

Notation declarations do not participate in [validation](#key-vn) as such. They are referenced in the course of [validating](#key-vn) strings as members of the [NOTATION](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#NOTATION) simple type.

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#n-annotation) property.

### XML Representation of Notation Declaration Schema Components

The XML representation for a notation declaration schema component is a [<notation>](#notation) element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

**XML Representation Summary:** notation **Element Information Item**

[Notation Declaration](#Notation_Declaration_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the parent attribute of the parent schema element information item.

{system identifier} The [actual value](#key-vv) of the system [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise attribute, if present, otherwise [absent](#key-null).

{public identifier} The [actual value](#key-vv) of the public [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element)attribute

{annotation} The annotation corresponding to the [<annotation>](#annotation) element information item in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if present, otherwise children, if present, otherwise [absent](#key-null).

**Example**

<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>

### Constraints on XML Representations of Notation Declarations

### Notation Declaration Validation Rules

None as such.

### Notation Declaration Information Set Contributions

### Constraints on Notation Declaration Schema Components

All notation declarations (see [Notation Declarations](#cNotation_Declarations)) must satisfy the following constraint.

## Annotations

Annotations provide for human- and machine-targeted annotations of schema components.

**Example**

<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.

### The Annotation Schema Component

The annotation schema component has the following properties:

**Schema Component:** [Annotation](#Annotation)

{application information} A sequence of element information items.

{user information} A sequence of element information items.

{attributes} A sequence of attribute information items.

[{user information}](#user_information) is intended for human consumption, [{application information}](#application_information) 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](#key-vn) does not involve dereferencing these URIs, when present. In the case of [{user information}](#user_information), indication should be given as to the identity of the (human) language used in the contents, using the xml:lang attribute.

[{attributes}](#attributes) 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](#key-vn) as such. Provided an annotation itself satisfies all relevant [Schema Component Constraints](#gloss-cos) it cannot affect the [validation](#key-vn) of element information items.

### 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 [<annotation>](#annotation) element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

**XML Representation Summary:** annotation **Element Information Item**

**XML Representation Summary:** appinfo **Element Information Item**

**XML Representation Summary:** documentation **Element Information Item**

[Annotation](#Annotation_details) **Schema Component**

**Property Representation**

{application information} A sequence of the [<appinfo>](#appinfo) element information items from among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), in order, if any, otherwise the empty sequence.children, in order, if any, otherwise the empty sequence.

{user information} A sequence of the [<documentation>](#documentation) element information items from among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), in order, if any, otherwise the empty sequence.children, in order, if any, otherwise the empty sequence.

{attributes} A sequence of attribute information items, namely those allowed by the attribute wildcard in the type definition for the [<annotation>](#annotation) 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 [<annotation>](#annotation) element in the example above will have one element item in each of its [{user information}](#user_information) and [{application information}](#application_information) and one attribute item in its [{attributes}](#attributes).

### Constraints on XML Representations of Annotations

### Annotation Validation Rules

None as such.

### Annotation Information Set Contributions

None as such: the addition of annotations to the [post-schema-validation infoset](#key-psvi) is covered by the [post-schema-validation infoset](#key-psvi) contributions of the enclosing components.

### Constraints on Annotation Schema Components

All annotations (see [Annotations](#cAnnotations)) must satisfy the following constraint.

## Simple Type Definitions

**Note**

This section consists of a combination of non-normative versions of normative material from [[XML Schemas: Datatypes]](#ref-xsp2), 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](http://www.w3.org/TR/xml-infoset/#infoitem.element) of element and attribute information items.children of element and attribute information items.

**Example**

<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.

### (non-normative) The Simple Type Definition Schema Component

The simple type definition schema component has the following properties:

**Schema Component:** [Simple Type Definition](#Simple_Type_Definition)

{name} Optional. An NCName as defined by [[XML-Namespaces]](#ref-xml-namespaces).

{target namespace} Either [absent](#key-null) or a namespace name, as defined in [[XML-Namespaces]](#ref-xml-namespaces).

{base type definition} A simple type definition, which may be the [simple ur-type definition](#simple-ur-type-itself).

{facets} A set of constraining facets.

{fundamental facets} A set of fundamental facets.

{final} A subset of {extension, list, restriction, union}.

{variety} One of {atomic, list, union}. Depending on the value of [{variety}](#variety), further properties are defined as follows:

**atomic**

{primitive type definition} A built-in primitive simple type definition.

**list**

{item type definition} A simple type definition.

**union**

{member type definitions} A non-empty sequence of simple type definitions.

{annotation} Optional. An annotation.

Simple types are identified by their [{name}](#st-name) and [{target namespace}](#st-target_namespace). Except for anonymous simple types (those with no [{name}](#st-name)), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an [XML Schema](#key-schema), no simple type definition can have the same name as another simple or complex type definition. Simple type [{name}](#st-name)s and [{target namespace}](#ct-target_namespace)s are provided for reference from instances (see [xsi:type](#xsi_type)), and for use in the XML representation of schema components (specifically in [<element>](#element) and [<attribute>](#attribute)). See [References to schema components across namespaces](#composition-schemaImport) for the use of component identifiers when importing one schema into another.

**Note**

The [{name}](#st-name) of a simple type is not ipso facto the [(local) name](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the element or attribute information items (local) name of the element or attribute information items [validated](#key-vn) by that definition. The connection between a name and a type definition is described in [Element Declarations](#cElement_Declarations) and [Attribute Declarations](#cAttribute_Declarations).

A simple type definition with an empty specification for [{final}](#st-final) can be used as the [{base type definition}](#st-base_type_definition) for other types derived by either of extension or restriction, or as the [{item type definition}](#st-item_type_definition) in the definition of a list, or in the [{member type definitions}](#st-member_type_definitions) 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.

[{variety}](#variety) determines whether the simple type corresponds to an atomic, list or union type as defined by [[XML Schemas: Datatypes]](#ref-xsp2).

As described in [Type Definition Hierarchy](#Type_Derivation), every simple type definition is a [restriction](#key-typeRestriction) of some other simple type (the [{base type definition}](#st-base_type_definition)), which is the [simple ur-type definition](#key-simpleUrType) 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 [{primitive type definition}](#st-primitive_type_definition).

[{facets}](#facets) for each simple type definition are selected from those defined in [[XML Schemas: Datatypes]](#ref-xsp2). For atomic definitions, these are restricted to those appropriate for the corresponding [{primitive type definition}](#st-primitive_type_definition). Therefore, the value space and lexical space (i.e. what is [validated](#key-vn) by any atomic simple type) is determined by the pair ([{primitive type definition}](#st-primitive_type_definition), [{facets}](#facets)).

As specified in [[XML Schemas: Datatypes]](#ref-xsp2), list simple type definitions [validate](#key-vn) space separated tokens, each of which conforms to a specified simple type definition, the [{item type definition}](#st-item_type_definition). The item type specified must not itself be a list type, and must be one of the types identified in [[XML Schemas: Datatypes]](#ref-xsp2) as a suitable item type for a list simple type. In this case the [{facets}](#facets) apply to the list itself, and are restricted to those appropriate for lists.

A union simple type definition [validates](#key-vn) strings which satisfy at least one of its [{member type definitions}](#st-member_type_definitions). As in the case of list, the [{facets}](#facets) apply to the union itself, and are restricted to those appropriate for unions.

As discussed in [Type Definition Hierarchy](#Type_Derivation), the [ur-type definition](#key-urType) functions as a simple type when used as the [base type definition](#st-base_type_definition) for the built-in primitive datatypes and for list and union type definitions. It 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 [simple ur-type definition](#key-simpleUrType) must not be named as the [base type definition](#st-base_type_definition) of any user-defined atomic simple type definitions: as it has no constraining facets, this would be incoherent.

See [Annotations](#cAnnotations) for information on the role of the [{annotation}](#st-annotation) property.

### (non-normative) XML Representation of Simple Type Definition Schema Components

**Note**

This section reproduces a version of material from [[XML Schemas: Datatypes]](#ref-xsp2), for local cross-reference purposes.

**XML Representation Summary:** simpleType **Element Information Item**

**XML Representation Summary:** restriction **Element Information Item**

**XML Representation Summary:** list **Element Information Item**

**XML Representation Summary:** union **Element Information Item**

[Simple Type Definition](#Simple_Type_Definition_details) **Schema Component**

**Property Representation**

{name} The [actual value](#key-vv) of the name [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) if present, otherwise attribute if present, otherwise [absent](#key-null).

{target namespace} The [actual value](#key-vv) of the targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the attribute of the [<schema>](#schema) ancestor element information item if present, otherwise [absent](#key-null).

{base type definition}

the [<restriction>](#restriction) alternative is chosen

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the base [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of attribute of [<restriction>](#restriction), if present, otherwise the type definition corresponding to the [<simpleType>](#simpleType) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction).

the [<list>](#list) or [<union>](#union) alternative is chosen

the [simple ur-type definition](#simple-ur-type-itself).

{final} As for the [{prohibited substitutions}](#ct-exact) property of complex type definitions, but using the final and finalDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) in place of the attributes in place of the block and blockDefault [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) and with the relevant set being attributes and with the relevant set being {extension, restriction, list, union}.

{variety} If the [<list>](#list) alternative is chosen, then list, otherwise if the [<union>](#union) alternative is chosen, then union, otherwise (the [<restriction>](#restriction) alternative is chosen), then the [{variety}](#variety) of the [{base type definition}](#st-base_type_definition).

If the [{variety}](#variety) is atomic, the following additional property mappings also apply:

[Atomic Simple Type Definition](#Simple_Type_Definition_details) **Schema Component**

**Property Representation**

{primitive type definition} The built-in primitive type definition from which the [{base type definition}](#st-base_type_definition) is derived.

{facets} A set of facet components [constituting a restriction](#key-facets-restriction) of the [{facets}](#facets) of the [{base type definition}](#st-base_type_definition) with respect to a set of facet components corresponding to the appropriate element information items among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) (i.e. those which specify facets, if any), as defined in [Simple Type Restriction (Facets)](#st-restrict-facets).

If the [{variety}](#variety) is list, the following additional property mappings also apply:

[List Simple Type Definition](#Simple_Type_Definition_details) **Schema Component**

**Property Representation**

{item type definition}

the [<list>](#list) alternative is chosen

the type definition [resolved](#src-resolve) to by the [actual value](#key-vv) of the itemType [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) of attribute of [<list>](#list), if present, otherwise the type definition corresponding to the [<simpleType>](#simpleType) among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<list>](#list).

the [<restriction>](#restriction) option is chosen

the [{item type definition}](#st-item_type_definition) of the [{base type definition}](#st-base_type_definition).

{facets} If the [<restriction>](#restriction) alternative is chosen, a set of facet components [constituting a restriction](#key-facets-restriction) of the [{facets}](#facets) of the [{base type definition}](#st-base_type_definition) with respect to a set of facet components corresponding to the appropriate element information items among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) (i.e. those which specify facets, if any), as defined in [Simple Type Restriction (Facets)](#st-restrict-facets), otherwise the empty set.

If the [{variety}](#variety) is union, the following additional property mappings also apply:

[Union Simple Type Definition](#Simple_Type_Definition_details) **Schema Component**

**Property Representation**

{member type definitions}

the [<union>](#union) alternative is chosen

[Definition:] define the explicit members as the type definitions [resolved](#src-resolve) to by the items in the [actual value](#key-vv) of the memberTypes [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, followed by the type definitions corresponding to the attribute, if any, followed by the type definitions corresponding to the [<simpleType>](#simpleType)s among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<union>](#union), if any. The actual value is then formed by replacing any union type definition in the [explicit members](#key-exm) with the members of their [{member type definitions}](#st-member_type_definitions), in order.

the [<restriction>](#restriction) option is chosen

the [{member type definitions}](#st-member_type_definitions) of the [{base type definition}](#st-base_type_definition).

{facets} If the [<restriction>](#restriction) alternative is chosen, a set of facet components [constituting a restriction](#key-facets-restriction) of the [{facets}](#facets) of the [{base type definition}](#st-base_type_definition) with respect to a set of facet components corresponding to the appropriate element information items among the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) of children of [<restriction>](#restriction) (i.e. those which specify facets, if any), as defined in [Simple Type Restriction (Facets)](#st-restrict-facets), otherwise the empty set.

### Constraints on XML Representations of Simple Type Definitions

### Simple Type Definition Validation Rules

### Simple Type Definition Information Set Contributions

None as such.

### Constraints on Simple Type Definition Schema Components

All simple type definitions other than the [simple ur-type definition](#simple-ur-type-itself) and the built-in primitive datatype definitions (see [Simple Type Definitions](#Simple_Type_Definitions)) must satisfy both the following constraints.

The following constraint defines relations appealed to elsewhere in this specification.

**Note**

With respect to [no-identity](#no-identity) above.

### Built-in Simple Type Definition

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

The [simple ur-type definition](#key-simpleUrType) 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 [{base type definition}](#st-base_type_definition) properties, and the [ur-type definition](#key-urType), which is its [{base type definition}](#st-base_type_definition). This is why the [simple ur-type definition](#simple-ur-type-itself) is exempted from the first clause of [Simple Type Definition Properties Correct](#st-props-correct), which would otherwise bar it because of its derivation from a complex type definition and absence of [{variety}](#variety).

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](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#built-in-primitive-datatypes) section of [[XML Schemas: Datatypes]](#ref-xsp2)) are present by definition in every schema. All are in the XML Schema [{target namespace}](#st-target_namespace) (namespace name http://www.w3.org/2001/XMLSchema), have an atomic [{variety}](#variety) with an empty [{facets}](#facets) and the [simple ur-type definition](#key-simpleUrType) as their [base type definition](#st-base_type_definition) and themselves as [{primitive type definition}](#st-primitive_type_definition).

Similarly, simple type definitions for all the built-in derived datatypes (see the [Derived Datatypes](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#built-in-derived) section of [[XML Schemas: Datatypes]](#ref-xsp2)) are present by definition in every schema, with properties as specified in [[XML Schemas: Datatypes]](#ref-xsp2) and as represented in XML in [Schema for Schemas (normative)](#normative-schemaSchema).

## Schemas as a Whole

A schema consists of a set of schema components.

**Example**

<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.

### The Schema Itself

At the abstract level, the schema itself is just a container for its components.

**Schema Component:** [Schema](#key-schema)

{type definitions} A set of named simple and complex type definitions.

{attribute declarations} A set of named (top-level) attribute declarations.

{element declarations} A set of named (top-level) element declarations.

{attribute group definitions} A set of named attribute group definitions.

{model group definitions} A set of named model group definitions.

{notation declarations} A set of notation declarations.

{annotations} A set of annotations.

### XML Representations of Schemas

A schema is represented in XML by one or more [schema documents](#key-schemaDoc), that is, one or more [<schema>](#schema) element information items. A [schema document](#key-schemaDoc) contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. A [schema document](#key-schemaDoc) which has one or more [<import>](#import) element information items corresponds to a schema with components with more than one target namespace, see [Import Constraints and Semantics](#src-import).

**XML Representation Summary:** schema **Element Information Item**

[Schema](#key-schema) **Schema Component**

**Property Representation**

{type definitions} The simple and complex type definitions corresponding to all the [<simpleType>](#simpleType) and [<complexType>](#complexType) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported definitions, see children, if any, plus any included or imported definitions, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{attribute declarations} The (top-level) attribute declarations corresponding to all the [<attribute>](#attribute) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported declarations, see children, if any, plus any included or imported declarations, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{element declarations} The (top-level) element declarations corresponding to all the [<element>](#element) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported declarations, see children, if any, plus any included or imported declarations, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{attribute group definitions} The attribute group definitions corresponding to all the [<attributeGroup>](#attributeGroup) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported definitions, see children, if any, plus any included or imported definitions, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{model group definitions} The model group definitions corresponding to all the [<group>](#group) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported definitions, see children, if any, plus any included or imported definitions, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{notation declarations} The notation declarations corresponding to all the [<notation>](#notation) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any, plus any included or imported declarations, see children, if any, plus any included or imported declarations, see [Assembling a schema for a single target namespace from multiple schema definition documents](#compound-schema) and [References to schema components across namespaces](#composition-schemaImport).

{annotations} The annotations corresponding to all the [<annotation>](#annotation) element information items in the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), if any.children, if any.

Note that none of the attribute information items displayed above correspond directly to properties of schemas. The blockDefault, finalDefault, attributeFormDefault, elementFormDefaultand 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 [XML Schema Abstract Data Model](#concepts-data-model) makes clear that most components have a target namespace. Most components corresponding to representations within a given [<schema>](#schema) 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](#key-schema) whose components have no [{target namespace}](#e-target_namespace) is one which has no targetNamespace attribute specified at all.

**Note**

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, [<schema>](#schema) 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 [<include>](#include) and [<import>](#import), which do not correspond directly to any schema component at all, each of the element information items which may appear in the content of [<schema>](#schema) corresponds to a schema component, and all except [<annotation>](#annotation) are named. The sections below present each such item in turn, setting out the components to which it may correspond.

#### 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 ([References to schema components across namespaces](#composition-schemaImport)) from an external schema (which may, but need not, correspond to an actual schema document). The form of all such references is a [QName](#gloss-QName).

[Definition:] A QName is a name with an optional namespace qualification, as defined in [[XML-Namespaces]](#ref-xml-namespaces). When used in connection with the XML representation of schema components or references to them, this refers to the simple type [QName](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#QName) as defined in [[XML Schemas: Datatypes]](#ref-xsp2).

[Definition:] An NCName is a name with no colon, as defined in [[XML-Namespaces]](#ref-xml-namespaces). When used in connection with the XML representation of schema components in this specification, this refers to the simple type [NCName](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#NCName) as defined in [[XML Schemas: Datatypes]](#ref-xsp2).

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.

**Example**

<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 [<complexType>](#complexType) 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 [References to schema components across namespaces](#composition-schemaImport) for a discussion of importing.

#### References to Schema Components from Elsewhere

The names of schema components such as type definitions and element declarations are not of type [ID](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#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]](#ref-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](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#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.

### Constraints on XML Representations of Schemas

[Definition:] Whenever the word resolve in any form is used in this chapter in connection with a [QName](#gloss-QName) in a schema document, the following definition [QName resolution (Schema Document)](#src-resolve) should be understood:

### Validation Rules for Schemas as a Whole

As the discussion above at [Schema Component Details](#components) makes clear, at the level of schema components and [validation](#key-vn), reference to components by name is normally not involved. In a few cases, however, qualified names appearing in information items being [validated](#key-vn) must be resolved to schema components by such lookup. The following constraint is appealed to in these cases.

### Schema Information Set Contributions

### Constraints on Schemas as a Whole

All schemas (see [Schemas as a Whole](#Schemas)) must satisfy the following constraint.

# Schemas and Namespaces: Access and Composition

This chapter defines the mechanisms by which this specification establishes the necessary precondition for [assessment](#key-va), 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.

[Conformance](#concepts-conformance) describes three levels of conformance for schema processors, and [Schemas and Schema-validity Assessment](#conformance) provides a formal definition of [assessment](#key-va). This section sets out in detail the 3-layer architecture implied by the three conformance levels. The layers are:

The [assessment](#key-va) core, relating schema components and instance information items;

Schema representation: the connections between XML representations and schema components, including the relationships between namespaces and schema components;

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](#key-va) of an instance element information item. Layer 2 specifies the use of [<schema>](#schema) 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.

## Layer 1: Summary of the Schema-validity Assessment Core

The fundamental purpose of the [assessment](#key-va) core is to define [assessment](#key-va) 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](#key-va) is defined with reference to an [XML Schema](#key-schema) (note not a [schema document](#key-schemaDoc)) which consists of (at a minimum) the set of schema components (definitions and declarations) required for that [assessment](#key-va). 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](#key-va) 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](#key-va) 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](#key-component) transitively required to complete an [assessment](#key-va) (note that components derived from [schema documents](#key-schemaDoc) 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](#key-va) to differ from that which would have been obtained from the same schema components acquired in bulk.

**Note**

the [assessment](#key-va) core is defined in terms of schema components at the abstract level, and no mention is made of the schema definition syntax (i.e. [<schema>](#schema)). 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](#key-va) core is concerned is to implement one or more of the options for [assessment](#key-va) given below in [Assessing Schema-Validity](#validation_outcome). Neither the choice of element information item for that [assessment](#key-va), nor which of the means of initiating [assessment](#key-va) are used, is within the scope of this specification.

Although [assessment](#key-va) 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](#key-va) outcome that is the same as would be given if [assessment](#key-va) was undertaken again with the final, fully assembled schema.

## Layer 2: Schema Documents, Namespaces and Composition

The sub-sections of [Schema Component Details](#components) 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](#key-va) 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.

**Note**

The core [assessment](#key-va) 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 [Conformance](#concepts-conformance), 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.

**Note**

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.

### 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](#key-schemaDoc), that is several [<schema>](#schema) element information items:

**XML Representation Summary:** include **Element Information Item**

A [<schema>](#schema) information item may contain any number of [<include>](#include) elements. Their schemaLocation attributes, consisting of a URI reference, identify other [schema documents](#key-schemaDoc), that is [<schema>](#schema) information items.

The [XML Schema](#key-schema) corresponding to [<schema>](#schema) contains not only the components corresponding to its definition and declaration [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), but also all the components of all the children, but also all the components of all the [XML Schemas](#key-schema) corresponding to any [<include>](#include)d schema documents. Such included schema documents must either (a) have the same targetNamespace as the [<include>](#include)ing schema document, or (b) no targetNamespace at all, in which case the [<include>](#include)d schema document is converted to the [<include>](#include)ing schema document's targetNamespace.

**Note**

The above is carefully worded so that multiple [<include>](#include)ing of the same schema document will not constitute a violation of [Schema Properties Correct](#sch-props-correct), but applications are allowed, indeed encouraged, to avoid [<include>](#include)ing the same schema document more than once to forestall the necessity of establishing identity component by component.

### 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.

**XML Representation Summary:** redefine **Element Information Item**

A [<schema>](#schema) information item may contain any number of [<redefine>](#redefine) elements. Their schemaLocation attributes, consisting of a URI reference, identify other [schema documents](#key-schemaDoc), that is [<schema>](#schema) information items.

The [XML Schema](#key-schema) corresponding to [<schema>](#schema) contains not only the components corresponding to its definition and declaration [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), but also all the components of all the children, but also all the components of all the [XML Schemas](#key-schema) corresponding to any [<redefine>](#redefine)d schema documents. Such schema documents must either (a) have the same targetNamespace as the [<redefine>](#redefine)ing schema document, or (b) no targetNamespace at all, in which case the [<redefine>](#redefine)d schema document is converted to the [<redefine>](#redefine)ing schema document's targetNamespace.

The definitions within the [<redefine>](#redefine) element itself are restricted to be redefinitions of components from the [<redefine>](#redefine)d schema document, in terms of themselves. That is, Not all the components of the [<redefine>](#redefine)d schema document need be redefined.

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 [<redefine>](#redefine)d selves.

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.

**Note**

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.

**Example**

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.

**Note**

The above is carefully worded so that multiple equivalent [<redefine>](#redefine)ing of the same schema document will not constitute a violation of [Schema Properties Correct](#sch-props-correct), but applications are allowed, indeed encouraged, to avoid [<redefine>](#redefine)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).

### References to schema components across namespaces

As described in [XML Schema Abstract Data Model](#concepts-data-model), 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:

**XML Representation Summary:** import **Element Information Item**

The [<import>](#import) element information item identifies namespaces used in external references, i.e. those whose [QName](#gloss-QName) identifies them as coming from a different namespace (or none) than the enclosing schema document's targetNamespace. The [actual value](#key-vv) of its namespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) 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 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](#key-schemaDoc); the processor is free to access or construct components using means of its own choosing.

The [actual value](#key-vv) of the schemaLocation, if present, gives a hint as to where a serialization of a [schema document](#key-schemaDoc) with declarations and definitions for that namespace (or none) may be found. When no schemaLocation [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is present, the schema author is leaving the identification of that schema to the instance, application or user, via the mechanisms described below in 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 [Layer 3: Schema Document Access and Web-interoperability](#composition-instances). 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](#key-schemaDoc) containing the component(s) in the [<import>](#import)ed namespace referred to elsewhere in the containing schema document.

**Note**

Since both the namespace and schemaLocation [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) are optional, a bare 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.

**Example**

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](#gloss-QName) 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.

**Note**

The above is carefully worded so that multiple [<import>](#import)ing of the same schema document will not constitute a violation of [Schema Properties Correct](#sch-props-correct), but applications are allowed, indeed encouraged, to avoid [<import>](#import)ing the same schema document more than once to forestall the necessity of establishing identity component by component. Given that the schemaLocation [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element) is only a hint, it is open to applications to ignore all but the first attribute is only a hint, it is open to applications to ignore all but the first [<import>](#import) for a given namespace, regardless of the [actual value](#key-vv) of schemaLocation, but such a strategy risks missing useful information when new schemaLocations are offered.

## Layer 3: Schema Document Access and Web-interoperability

Layers 1 and 2 provide a framework for [assessment](#key-va) 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.

### Standards for representation of schemas and retrieval of schema documents on the Web

For interoperability, serialized [schema documents](#key-schemaDoc), 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 [<schema>](#schema) element information items).

**Note**

there will often be times when a schema document will be a complete XML 1.0 document whose document element is [<schema>](#schema). There will be other occasions in which [<schema>](#schema) items will be contained in other documents, perhaps referenced using fragment and/or XPointer notation.

**Note**

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](#key-schemaDoc). An Accept header of application/xml, text/xml; q=0.9, \*/\* is perhaps a reasonable starting point.

### How schema definitions are located on the Web

As described in [Layer 1: Summary of the Schema-validity Assessment Core](#layer1), processors are responsible for providing the schema components (definitions and declarations) needed for [assessment](#key-va). This section introduces a set of normative conventions to facilitate interoperability for instance and schema documents retrieved and processed from the Web.

**Note**

As discussed above in [Layer 2: Schema Documents, Namespaces and Composition](#layer2), other non-Web mechanisms for delivering schemas for [assessment](#key-va) may exist, but are outside the scope of this specification.

Processors on the Web are free to undertake [assessment](#key-va) against arbitrary schemas in any of the ways set out in [Assessing Schema-Validity](#validation_outcome). 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](#key-va) of a document on the web must behave as follows:

unless directed otherwise by the user, [assessment](#key-va) 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](#key-va) is undertaken.

The composition of the complete schema for use in [assessment](#key-va) is discussed in [Layer 2: Schema Documents, Namespaces and Composition](#layer2) above. The means used to locate appropriate schema document(s) are processor and application dependent, subject to the following requirements:

Schemas are represented on the Web in the form specified above in [Standards for representation of schemas and retrieval of schema documents on the Web](#schema-repr);

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.

**Note**

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.

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](http://www.w3.org/TR/xml-infoset/#infoitem.element) (in the XML Schema instance namespace, that is, 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 targetNamespace [attribute](http://www.w3.org/TR/xml-infoset/#infoitem.element).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](#key-vv) of such xsi:schemaLocation and xsi:noNamespaceSchemaLocation [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element), see details below.attributes, see details below.

xsi:schemaLocation and xsi:noNamespaceSchemaLocation [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) can occur on any element. However, it is an error if such an attribute occurs 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](#key-vn) whose [namespace name](http://www.w3.org/TR/xml-infoset/#infoitem.attribute) it addresses. According to the rules of namespace name it addresses. According to the rules of [Layer 1: Summary of the Schema-validity Assessment Core](#layer1), the corresponding schema may be lazily assembled, but is otherwise stable throughout [assessment](#key-va). Although schema location attributes can occur on any element, and can be processed incrementally as discovered, their effect is essentially global to the [assessment](#key-va). Definitions and declarations remain in effect beyond the scope of the element on which the binding is declared.

**Example**

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.

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.

# 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.

## Errors in Schema Construction and Structure

Before [assessment](#key-va) can be attempted, a schema is required. Special-purpose applications are free to determine a schema for use in [assessment](#key-va) by whatever means are appropriate, but general purpose processors should implement the strategy set out in [Schema Document Location Strategy](#schema_reference), starting with the namespaces declared in the document whose [assessment](#key-va) is being undertaken, and the [actual value](#key-vv)s of the xsi:schemaLocation and xsi:noNamespaceSchemaLocation [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element) thereof, if any, along with any other information about schema identity or schema document location provided by users in application-specific ways, if any.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 [Schema Component Details](#components).

If a schema is derived from one or more schema documents (that is, one or more [<schema>](#schema) element information items) based on the correspondence rules set out in [Schema Component Details](#components) and [Schemas and Namespaces: Access and Composition](#composition), 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 [Schema for Schemas (normative)](#normative-schemaSchema), that is, following schema-validation with such a schema, the [<schema>](#schema) element information items would have a [{validation attempted}](#e-validation_attempted) property with value full or partial and a [{validity}](#e-validity) 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 [Schema Representation Constraints](#outcome-src).

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](#key-va) 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](#key-va) using such non-schemas.

## Assessing Schema-Validity

With a schema which satisfies the conditions expressed in [Errors in Schema Construction and Structure](#conformance-schemaValidity) above, the schema-validity of an element information item can be assessed. Three primary approaches to this are possible:

The user or application identifies a complex type definition from among the [{type definitions}](#type_definitions) of the schema, and appeals to [Schema-Validity Assessment (Element)](#cvc-assess-elt) (

The user or application identifies a element declaration from among the [{element declarations}](#element_declarations) of the schema, checks that its [{name}](#e-name) and [{target namespace}](#e-target_namespace) match the [local name](http://www.w3.org/TR/xml-infoset/#infoitem.element) and local name and [namespace name](http://www.w3.org/TR/xml-infoset/#infoitem.element) of the item, and appeals to namespace name of the item, and appeals to [Schema-Validity Assessment (Element)](#cvc-assess-elt) (

The processor starts from [Schema-Validity Assessment (Element)](#cvc-assess-elt) with no stipulated declaration or definition, and either [strict](#key-sva) or [lax](#key-lva) 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](http://www.w3.org/TR/xml-infoset/#infoitem.element) and attributes and [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), recursively, as defined by children, recursively, as defined by [Assessment Outcome (Element)](#sic-e-outcome) and [Assessment Outcome (Attribute)](#sic-a-outcome). It is up to applications to decide what constitutes a successful outcome.

Note that every element and attribute information item participating in the [assessment](#key-va) will also have a validation context property which refers back to the element information item at which [assessment](#key-va) began. [Definition:] This item, that is the element information item at which [assessment](#key-va) began, is called the validation root.

**Note**

This specification does not reconstruct the XML 1.0 notion of root in either schemas or instances. Equivalent functionality is provided for at [assessment](#key-va) invocation, via

**Note**

This specification has nothing normative to say about multiple [assessment](#key-va) episodes. It should however be clear from the above that if a processor restarts [assessment](#key-va) with respect to a [post-schema-validation infoset](#key-psvi) some [post-schema-validation infoset](#key-psvi) contributions from the previous [assessment](#key-va) 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](#key-va) was not attempted because of a [validation](#key-vn) failure, but declarations and/or definitions are available for at least some of the [children](http://www.w3.org/TR/xml-infoset/#infoitem.element) or children or [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element);attributes;

[assessment](#key-va) was not attempted because a named definition or declaration was missing, but after further effort the processor has retrieved it.

[assessment](#key-va) 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](http://www.w3.org/TR/xml-infoset/#infoitem.element) or children or [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element).attributes.

## Missing Sub-components

At the beginning of [Schema Component Details](#components), 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](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#QName), and the [resolution](#src-resolve) of such [QNames](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#QName) may fail, resulting in one or more values of or containing [absent](#key-null) where a component is mandated.

If at any time during [assessment](#key-va), an element or attribute information item is being [validated](#key-vn) with respect to a component of any kind any of whose properties has or contains such an [absent](#key-null) value, the [validation](#key-vn) is modified, as following:

In the case of attribute information items, the effect is as if [Attribute Locally Valid](#cvc-attribute) had failed;

In the case of element information items, the effect is as if [Element Locally Valid (Element)](#cvc-elt) had failed;

In the case of element information items, processors may choose to continue [assessment](#key-va): see [lax assessment](#key-lva).

Because of the value specification for [{validation attempted}](#e-validation_attempted) in [Assessment Outcome (Element)](#sic-e-outcome), if this situation ever arises, the document as a whole cannot show a [{validation attempted}](#e-validation_attempted) of full.

## 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 [Conformance](#concepts-conformance)) they support, consistently with the conditions set out above.

# 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.

**Note**

And that is the end of the schema for schema documents.

# 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>

# Outcome Tabulations (normative)

To facilitate consistent reporting of schema errors and [validation](#key-vn) 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 [Derivation Valid (Extension)](#cos-ct-extends).

## Validation Rules

## Contributions to the post-schema-validation infoset

## Schema Representation Constraints

## Schema Component Constraints

# Required Information Set Items and Properties (normative)

This specification requires as a precondition for [assessment](#key-va) an information set as defined in [[XML-Infoset]](#ref-xmlinfo) which supports at least the following information items and properties:

**Attribute Information Item**

[local name](http://www.w3.org/TR/xml-infoset/#infoitem.attribute), local name, [namespace name](http://www.w3.org/TR/xml-infoset/#infoitem.attribute), namespace name, [normalized value](http://www.w3.org/TR/xml-infoset/#infoitem.attribute)normalized value

**Character Information Item**

[character code](http://www.w3.org/TR/xml-infoset/#infoitem.character)character code

**Element Information Item**

[local name](http://www.w3.org/TR/xml-infoset/#infoitem.element), local name, [namespace name](http://www.w3.org/TR/xml-infoset/#infoitem.element), namespace name, [children](http://www.w3.org/TR/xml-infoset/#infoitem.element), children, [attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element), attributes, [in-scope namespaces](http://www.w3.org/TR/xml-infoset/#infoitem.element) or in-scope namespaces or [namespace attributes](http://www.w3.org/TR/xml-infoset/#infoitem.element)namespace attributes

**Namespace Information Item**

[prefix](http://www.w3.org/TR/xml-infoset/#infoitem.namespace), prefix, [namespace name](http://www.w3.org/TR/xml-infoset/#infoitem.namespace)namespace name

In addition, infosets should support the [unparsedEntities](http://www.w3.org/TR/xml-infoset/#infoitem.document) property of the Document Information Item. Failure to do so will mean all items of type unparsedEntities property of the Document Information Item. Failure to do so will mean all items of type [ENTITY](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#ENTITY) or [ENTITIES](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#ENTITIES) will fail to [validate](#key-vn).

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](http://www.w3.org/TR/2001/WD-xml-infoset-20010316/#conformance) to the [[XML-Infoset]](#ref-xmlinfo) specification.

# Schema Components Diagram (non-normative)

# 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.

# 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.

# Analysis of the Unique Particle Attribution Constraint (non-normative)

A specification of the import of [Unique Particle Attribution](#cos-nonambig) which does not appeal to a processing model is difficult. What follows is intended as guidance, without claiming to be complete.

[Definition:] Two non-group particles overlap if or or or

They are both element declaration particles whose declarations have the same [{name}](#e-name) and [{target namespace}](#e-target_namespace).

They are both element declaration particles one of whose [{name}](#e-name) and [{target namespace}](#e-target_namespace) are the same as those of an element declaration in the other's [substitution group](#key-eq).

They are both wildcards, and the intensional intersection of their [{namespace constraint}](#namespace_constraint)s as defined in [Attribute Wildcard Intersection](#cos-aw-intersect) is not the empty set.

One is a wildcard and the other an element declaration, and the [{target namespace}](#e-target_namespace) of any member of its [substitution group](#key-eq) is [valid](#key-vn) with respect to the [{namespace constraint}](#namespace_constraint) of the wildcard.

A content model will violate the unique attribution constraint if it contains two particles which [overlap](#key-overlap) and which either or

are both in the [{particles}](#particles) of a choice or all group

may [validate](#key-vn) adjacent information items and the first has [{min occurs}](#p-min_occurs) less than [{max occurs}](#p-max_occurs).

Two particles may [validate](#key-vn) 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.

# 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/>

# Acknowledgements (non-normative)

The following contributed material to the first edition of this specification:

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
* 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
* 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
* 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 Beged-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 Arbouzov, 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
* 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