3.4 Complex Type Definitions

Complex Type Definitions provide for:

- Constraining element information items by providing Attribute Declaration (§2.2.2.3)s governing the appearance and content of [attributes]
- Constraining element information item [children] to be empty, or to conform to a specified element-only or mixed content model, or else constraining the character information item [children] to conform to a specified simple type definition.
- Constraining elements and attributes to exist, not to exist, or to have specified values, with Assertion (§2.2.4.2)s.
- Using the mechanisms of Type Definition Hierarchy (§2.2.1.1) to derive a complex type from another simple or complex type.
- Specifying post-schema-validation infoset contributions for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

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.

3.4.1 The Complex Type Definition Schema Component

A complex type definition schema component has the following properties:

{Word removed the table when I transferred the contents into it. I've not replaced it to see if it's possible to do without it.}

Schema Component: Complex Type Definition, a ~~kind~~ form of Type Definition

{I've moved the explanatory text into the properties to reduce the number of places where things are described.}

{annotations}

A sequence of Annotation components.

See Annotations (§3.14) for information on the role of the {annotations} property.

{name}

An xs:NCName value. Optional.

Complex type definitions are identified by their {name} and {target namespace}. Except for anonymous complex type definitions (which have no {name}), within a schema the {name} of each complex type definition MUST be unique from that of any other simple or complex type definition.

Complex type {name}s and {target namespace}s are provided for reference from instances (see xsi:type (§2.6.1)), and for use in the XML representation of schema components (specifically in <element>). See References to schema components across namespaces (`<import>`) (§4.2.3) for the use of component identifiers when importing one schema into another.

{target namespace}

An xs:anyURI value. Optional.

{base type definition}

A type definition component. Required.

As described in Type Definition Hierarchy (§2.2.1.1), each complex type is derived from a {base type definition} which is itself either a Simple Type Definition (§2.2.1.2) or a Complex Type Definition (§2.2.1.3).

{final}

A subset of {*substitution*, *extension*, *restriction*, *list*, *union*}.

A complex type with an empty specification for {final} can be used as a {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.

{context}

Required if {name} is ·absent·, otherwise MUST be ·absent·.

Either an Element Declaration or a Complex Type Definition.

The {context} property is only relevant for anonymous type definitions, for which its value is the Element Declaration or a Complex Type Definition in which this type definition appears ~~as the value of a property, e.g. {type definition}~~.

{derivation method}

One of {*extension*, *restriction*}. Required.

{derivation method} specifies the means of derivation as either ***extension*** or ***restriction*** (see Type Definition Hierarchy (§2.2.1.1)).

{abstract}

An xs:boolean value. Required.

Complex types for which {abstract} is ***true*** MUST NOT be used as the {type definition} for the ·validation· of element information items. It follows that they MUST NOT be referenced from an xsi:type (§2.6.1) attribute in an instance document. Abstract complex types can be used as {base type definition}s, or even as the {type definition}s of element declarations, provided in every case a concrete derived type definition is used for ·validation·, either via xsi:type (§2.6.1) or the operation of a substitution group.

{attribute uses}

A set of Attribute Use components.

{attribute uses} are a set of attribute uses. See Element Locally Valid (Complex Type) (§3.4.4) and Attribute Locally Valid (§3.2.4) for details of attribute ·validation·.

{attribute wildcard}

A Wildcard component. Optional.

{attribute wildcard}s provide a more flexible specification for ·validation· of attributes not explicitly included in {attribute uses}. See Element Locally Valid (Complex Type) (§3.4.4), The Wildcard Schema Component (§3.10.1) and Wildcard allows Expanded Name (§3.10.4) for formal details of attribute wildcard ·validation·.

{content type}

A ~~Content Type~~ property record consisting of the next three properties. Required.

{content type} determines the :validation: of [children] of element information items.

{I've used notation along the lines of {base type definition}.{content type}.{variety} to replace the more narrative way of describing properties within properties. Such notation would need to be described in the earlier sections (2.x.x) of the document.}

`{content type}.{variety}`

One of {*empty*, *simple*, *element-only*, *mixed*}. Required.

Informally:

- {content type}.{variety} ***empty*** :validates: elements with no character or element information item [children].
- {content type}.{variety} ***simple*** :validates: elements with character-only [children]using ~~its~~ {content type}.{simple type definition}.
- {content type}.{variety} ***element-only*** :validates: elements with [children] that conform to the :content model: supplied by ~~its~~ {content type}.{particle}.
- {content type}.{variety} ***mixed*** :validates: elements whose element [children] (i.e. specifically ignoring other [children] such as character information items) conform to the :content model: supplied by {content type}.{particle}.

`{content type}.{particle}`

A Particle component. Required if {variety} is ***element-only*** or ***mixed***, otherwise MUST be :absent:.

`{content type}.{simple type definition}`

A Simple Type Definition component. Required if {variety} is ***simple***, otherwise MUST be :absent:.

`{prohibited substitutions}`

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

{prohibited substitutions} determine whether an element declaration appearing in a : content model: is prevented from additionally :validating: element items with an xsi:type (§2.6.1) 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} is empty, then all such substitutions are allowed, otherwise, the derivation method(s) it names are disallowed.

{assertions}

A sequence of Assertion components.

{assertions} constrain elements and attributes to exist, not to exist, or to have specified values. Though specified as a sequence, the order among the assertions is not significant during assessment. See Assertions (§3.12).

3.4.2 XML Representation of Complex Type Definitions

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

The XML representation for complex type definitions with a {content type}.{variety} ***simple*** is significantly different from that of those with other {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

```
<complexType
  abstract = boolean : false
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
  id = ID
  mixed = boolean : false
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleContent | complexContent | ((group
| all | choice | sequence)?, ((attribute | attributeGroup)*,
anyAttribute?), (assert | report)*)))
</complexType>
```

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

Complex Type Definition **Schema Component**

Property Representation	
{name}	The ·actual value· of the `name` [attribute] if present, otherwise ·absent·.
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the <schema> ancestor element information item if present, otherwise ·absent·.
{abstract}	The ·actual value· of the `abstract` [attribute], if present, otherwise ***false***.

{prohibited substitutions}

A set corresponding to the :actual value: of the `block` [attribute], if present, otherwise on the :actual value: of the `blockDefault` [attribute] of the ancestor <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 appropriate **case** among the following:

1 **If** the **EBV** is the empty string, **then** the empty set;

2 **If** the **EBV** is `#all`, **then** {***extension***, ***restriction***};

3 **otherwise** a set with members drawn from the set above, each being present or absent depending on whether the :actual value: (which is a list) contains an equivalently named item.

Note: Although the `blockDefault` [attribute] of <schema> may include values other than ***restriction*** or***extension***, those values are ignored in the determination of {prohibited substitutions} for complex type definitions (they *are* used elsewhere).

{final}

As for {prohibited substitutions} above, but using the `final` and `finalDefault` [attributes] in place of the `block` and `blockDefault` [attributes].

{context}

If the `name` [attribute] is present, then :absent:, otherwise (the parent element information item will be <element>), the Element Declaration corresponding to that parent information item.

{assertions}

A sequence whose members are Assertions drawn from the following sources, in order:

1 The {base type definition}.{assertions}.

2 Assertions corresponding to all the <assert> and <report> element information items among the [children], if any, in order.

{annotations}

The annotations corresponding to the <annotation> element information item in the [children], if present, in the <simpleContent> and <complexContent> [children], if present, and in their <restriction> and <extension> [children], if present, otherwise :absent:.

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

```
<simpleContent
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | extension))
</simpleContent>
```

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?, (minExclusive | minInclusive
| maxExclusive | maxInclusive | totalDigits | fractionDigits | maxScale
| minScale | length | minLength | maxLength | enumeration |
whiteSpace | pattern)*)?, ((attribute | attributeGroup)*,
anyAttribute?), (assert | report)*)
</restriction>
<extension
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((attribute | attributeGroup)*,
anyAttribute?), (assert | report)*)
</extension>
<attributeGroup
  id = ID
  ref = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</attributeGroup>
<anyAttribute
  id = ID
  namespace = ((##any | ##other) | List of (anyURI |
(##targetNamespace | ##local)) )
  notNamespace = List of (anyURI | (##targetNamespace |
##local))
  notQName = List of QName
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</anyAttribute>
```

Complex Type Definition with simple content **Schema Component Property Representation**

{base type definition}

The type definition resolved to by the actual value of the `base` [attribute]

{derivation method}

If the <restriction> alternative is chosen, then ***restriction***, otherwise (the <extension> alternative is chosen) ***extension***.

{attribute uses}

A union of sets of attribute uses as follows

1 The set of attribute uses corresponding to the <attribute> [children], if any.
2 The {attribute uses} of the attribute groups :resolved: to by the :actual value:s of the `ref` [attribute] of the <attributeGroup> [children], if any.
3 if the {base type definition} is a complex type definition, the {attribute uses} of that type definition, unless the <restriction> alternative is chosen, in which case some members of that type definition's {attribute uses} MUST NOT be included, namely those whose {attribute declaration}'s {name} and {target namespace} are the same as **one** of the following:
3.1 the {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 or clause 2 above;
3.2 what would have been the {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 above but for the :actual value: of the `use` [attribute] of the relevant <attribute> among the [children] of <restriction> being ***prohibited***.

{attribute wildcard}

1 [Definition:] Let the **local wildcard** be defined as the appropriate **case** among the following:
1.1 **If** there is an <anyAttribute> present, **then** a wildcard based on the :actual value:s of its [attributes] and the <annotation> [children], exactly as for the wildcard corresponding to an <any> element as set out in XML Representation of Wildcard Schema Components (§3.10.2);
1.2 **otherwise** :absent:.
2 [Definition:] Let the **complete wildcard** be defined as the appropriate **case** among the following:
2.1 **If** there are no <attributeGroup> [children] corresponding to attribute groups with :non-absent: {attribute wildcard}s, **then** the :local wildcard:.
2.2 **If** there are one or more <attributeGroup> [children] corresponding to attribute groups with :non-absent: {attribute wildcard}s, **then** the appropriate **case** among the following:
2.2.1 **If** there is an <anyAttribute> present, **then** a wildcard whose {process contents} and {annotations} are those of the :local wildcard:, and whose {namespace constraint} is the intensional intersection of the {namespace constraint} of the :local wildcard: and of the {namespace constraint}s of all the :non-absent: {attribute wildcard}s of the attribute groups corresponding to the <attributeGroup> [children], as defined in Attribute Wildcard Intersection (§3.10.6).
2.2.2 **If** there is no <anyAttribute> present, **then** a wildcard whose properties are as follows:

{process contents}

The {process contents} of the first :non-absent: {attribute wildcard} of an attribute group among the attribute groups corresponding to the <attributeGroup> [children].

{namespace constraint}

The intensional intersection of the {namespace constraint}s of all the :non-absent: {attribute wildcard}s of the attribute groups corresponding to the

<attributeGroup> [children], as defined in Attribute Wildcard Intersection (§3.10.6).

{annotations}

:absent:.

3 The value is then determined by the appropriate **case** among the following:

3.1 **If** the <restriction> alternative is chosen, **then** the :complete wildcard:;

3.2 **If** the <extension> alternative is chosen, **then**

3.2.1 [Definition:] let the **base wildcard** be defined as the appropriate **case** among the following:

3.2.1.1 **If** the {base type definition} is a complex type definition with an {attribute wildcard}, **then** that {attribute wildcard}.

3.2.1.2 **otherwise** :absent:.

3.2.2 The value is then determined by the appropriate **case** among the following:

3.2.2.1 **If** the :base wildcard: is :non-absent: , **then** the appropriate **case** among the following:

3.2.2.1.1 **If** the :complete wildcard: is :absent:, **then** the :base wildcard:.

3.2.2.1.2 **otherwise** a wildcard whose {process contents} and {annotations} are those of the :complete wildcard:, and whose {namespace constraint} is the intensional union of the {namespace constraint} of the :complete wildcard: and of the :base wildcard:, as defined in Attribute Wildcard Union (§3.10.6).

3.2.2.2 **otherwise** (the :base wildcard: is :absent:) the :complete wildcard:

{content type}

A Content Type ~~as follows~~ with the following three properties:

{content type}.{variety}

simple

{content type}.{particle}

:absent:

{content type}.{simple type definition}

*{This section – and others – repeated use the phrase "*type definition *:resolved: to by the :actual value: of the* `base` *[attribute]". It's easier for the reader just to use the term* {base type definition}. *}*

{This section often says things like, "If X and Y and Z and you're using <restriction> then". It's easier for the reader to have the context first, eg. phrase as "If you're using <restriction> and X and Y and Z then."}

{I've defined a few additional terms to make this section easier to read.}

Defined as follows:

If present, define the *Local Simple Type* as the simple type definition corresponding to the <simpleType> among the [children] of <restriction>.

If present, define the *Restricting Facets* as the set of facet components corresponding to the appropriate element information items among the <restriction>'s [children] (i.e. those which specify facets, if any).

If the <restriction> alternative is chosen, define the *Unrestricted Type* as either the *Local Simple Type* if there is one or, if not, the {base type definition}.{content type}.{simple type definition}.

{content type}.{simple type definition} is then the appropriate case among the following:

1 **If** the <restriction> alternative is chosen and the {base type definition} is a complex type where {base type definition}.{content type}.{variety} is ***simple*** **then** the *Unrestricted Type* restricted by the *Restricting Facets* as per Simple Type Restriction (Facets) (§3.15.6);

2 **If** the <restriction> alternative is chosen and the {base type definition} is a complex type definition where {base type definition}.{content type}.{variety} is ***mixed*** and {base type definition}.{content type}.{particle} is ~~:~~emptiable~~:~~, as defined in Particle Emptiable (§3.9.6), **then** this property is a simple type definition which restricts the *Local Simple Type* (which MUST be present) with the *Restricting Facets* as per Simple Type Restriction (Facets) (§3.15.6);

3 **If** the <extension> alternative is chosen and the {base type definition} is a complex type definition (whose own {content type}.{variety} MUST be ***simple***~~, see below~~*{Seems too far below to be of use!}*), **then** this property is set to {base type definition}.{content type}.{simple type definition};

4 **If** the <extension> alternative is chosen and the {base type definition} is a simple type definition, **then** this property is set to {base type definition}.{content type}.{simple type definition};

5 **otherwise** An ERROR!!! (could be an attempt to restrict a simple type base!)

```
<complexContent
  id = ID
  mixed = boolean
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | extension))
</complexContent>
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (group | all | choice | sequence)?,
((attribute | attributeGroup)*, anyAttribute?), (assert | report)*)
</restriction>
<extension
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((group | all | choice | sequence)?,
((attribute | attributeGroup)*, anyAttribute?), (assert | report)*))
</extension>
```

Complex Type Definition with complex content **Schema Component Property Representation**

{base type definition}

The type definition -resolved- to by the -actual value- of the `base` [attribute].

{derivation method}

If the <restriction> alternative is chosen, then ***restriction***, otherwise (the <extension> alternative is chosen) ***extension***.

{attribute uses}

A union of sets of attribute uses as follows:

1 The set of attribute uses corresponding to the <attribute> [children], if any.

2 The {attribute uses} of the attribute groups -resolved- to by the -actual value-s of the `ref` [attribute] of the <attributeGroup> [children], if any.

3 The {base type definition}.{attribute uses}, unless the <restriction> alternative is chosen, in which case the set of {attribute declaration}s in {base type definition}.{attribute uses} excluding those whose {name} and {target namespace} are the same as **one** of the following: *{The following are re-worded}*

3.1 The {name} and {target namespace} of an {attribute declaration} whose attribute use is in the set per clause 1 or clause 2 above;

3.2 what would have been the {name} and {target namespace} of an {attribute declaration} whose attribute use is in the set per clause 1 above but for the -actual value- of the `use` [attribute] of the relevant <attribute> among the [children] of <restriction> being ***prohibited***.

{attribute wildcard}

As above for the <simpleContent> alternative.

{content type}

{For multiple nested clauses of 'one of these' and 'all of these' I've prepended the various sentences with 'Or' or 'And' to help the reader remember what is going on.}

1 [Definition:] Let the **effective mixed** be the appropriate **case** among the following:

1.1 **If** the `mixed` [attribute] is present on <complexContent>, **then** its -actual value-;

1.2 **otherwise** *{presumably the value on ComplexContent takes precedence – or is it an error for them to differ?}* **If** the `mixed` [attribute] is present on <complexType>, **then** its -actual value-;

1.3 **otherwise** `false`.

2 [Definition:] Let the **effective content** be the appropriate **case** among the following:

2.1 **If one** of the following is true

2.1.1 There is no <group>, <all>, <choice> or <sequence> among the [children];

2.1.2 Or there is an <all> or <sequence> among the [children] with no [children] of its own excluding <annotation>;

2.1.3 Or there is a <choice> among the [children] whose `minOccurs` [attribute] has the :actual value: `0` and has no [children] of its own excluding <annotation>;

, **then** the appropriate **case** among the following:

2.1.4 **If** the :effective mixed: is `true`, **then** A particle whose properties are as follows:

{min occurs}

`1`

{max occurs}

`1`

{term}

A model group whose {compositor} is ***sequence*** and whose {particles} is empty.

2.1.5 **otherwise** ***empty***

2.2 **otherwise** the particle corresponding to the <all>, <choice>, <group> or <sequence> among the [children].

3 Then the value of the property is the appropriate **case** among the following:

3.1 **If** the <restriction> alternative is chosen, **then** the appropriate **case** among the following:

3.1.1 **If** the :effective content: is ***empty*** , **then** a Content Type as follows:

{content type}.{variety} ***empty,*** {content type}.{particle} :absent: and {content type}.{simple type definition} :absent:.

3.1.2 **otherwise** Content Type has the following properties:

{content type}.{variety}

mixed if the :effective mixed: is `true`, otherwise ***elementOnly***

{content type}.{particle}

The :effective content:

{content type}.{simple type definition}

:absent:

3.2 **If** the <extension> alternative is chosen, **then** the appropriate **case** among the following:

3.2.1 **If** the :effective content: is ***empty***, **then** {base type definition}.{content type}

3.2.2 Or **If** {base type definition}.{content type}.{variety} is ***empty*** or ***simple***, **then** a Content Type as per clause 3.1.2 above;

3.2.3 **otherwise** Content Type has the following properties:

{content type}.{variety}

mixed if the :effective mixed: is `true`, otherwise ***elementOnly***

{content type}.{particle}

a Particle whose properties are as follows:

{min occurs}

`1`

{max occurs}

`1`

{term}

A model group whose {compositor} is ***sequence*** and whose {particles} are the particle of {base type definition}.{content type} followed by the effective content.

{content type}.{simple type definition}

absent

Note: If the {base type definition} is a complex type definition, then the {assertions} always contain members of the {assertions} of the {base type definition}, no matter which alternatives are chosen in the XML representation, <simpleContent> or <complexContent>, <restriction> or <extension>.

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

Note: The *only* substantive function of the value ***prohibited*** for the `use` attribute of an <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}. Such an <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 (§3.4.4) or Constraints on Complex Type Definition Schema Components (§3.4.6).

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:rootType||
```

```
     ">
   <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· 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· 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:decimal"/>
</xs:complexType>
```

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

3.4.3 Constraints on XML Representations of Complex Type Definitions

Schema Representation Constraint: Complex Type Definition Representation OK

In addition to the conditions imposed on <complexType> element information items by the schema for schemas, **all** of the following also apply:

1 If the <complexContent> alternative is chosen, {base type definition}.{content type} MUST be a complex type definition;

2 If the <simpleContent> alternative is chosen, **all** of the following MUST be true:

  2.1 One of the following MUST apply:

    2.1.1 {base type definition}.{content type} is a complex type definition and {base type definition}.{content type}.{variety} is ***simple***;

    2.1.2 Or, only if the <restriction> alternative is also chosen, {base type definition}.{content type} is a complex type definition and {base type definition}.{content type}.{variety} is ***mixed*** and {base type

definition}.{content type}.{particle} is a Particle which is ·emptiable·, as defined in Particle Emptiable (§3.9.6);

2.1.3 Or, only if the <extension> alternative is also chosen, {base type definition}.{content type} is a simple type definition.

2.2 If clause 2.1.2 above is satisfied, then there is a <simpleType> among the [children] of <restriction>.

Note: Although not explicitly ruled out either here or in Schema for Schema Documents (Structures) (normative) (§A), specifying `<xs:complexType . . .mixed='true'` when the <simpleContent> alternative is chosen has no effect on the corresponding component, and SHOULD be avoided. This may be ruled out in a subsequent version of this specification.

3 The corresponding complex type definition component MUST satisfy the conditions set out in Constraints on Complex Type Definition Schema Components (§3.4.6);

3.4.4 Complex Type Definition Validation Rules

Validation Rule: Element Locally Valid (Complex Type)

For an element information item to be locally ·valid· with respect to a complex type definition **all** of the following MUST be true:

1 If clause 3.2 of Element Locally Valid (Element) (§3.3.4) (regarding nillable content) *{Helps to give the reader a reminder what it is about}* did not apply, then the appropriate **case** among the following is true:

1.1 **If** {content type}.{variety} is ***empty***, **then** the element information item has no character or element information item [children].

1.2 **If** {content type}.{variety} is ***simple***, **then** the element information item has no element information item [children], and the ·normalized value· of the element information item is ·valid· with respect to the {content type}.{simple type definition} as defined by String Valid (§3.15.4).

1.3 **If** {content type}.{variety} is ***element-only***, **then** the element information item has no character information item [children] other than those whose [character code] is defined as a white space in [XML 1.1].

Note: It is implementation-defined whether a schema processor supports the definition of white space from [XML 1.1], or that from [XML 1.0], or both.

1.4 **If** {content type}.{variety} is ***element-only*** or ***mixed***, **then** the sequence of the element information item's element information item [children], if any, taken in order, is ·valid· with respect to the {content type}.{particle}, as defined in Element Sequence Locally Valid (Particle) (§3.9.4.2).

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

2.1 **If** there is among the {attribute uses} an attribute use with an {attribute declaration} whose {name} matches the attribute information item's [local

name] and whose {target namespace} is identical to the attribute information item's [namespace name] (where an ·absent· {target namespace} is taken to be identical to a [namespace name] with no value), **then** the attribute information is ·valid· with respect to that attribute use as per Attribute Locally Valid (Use) (§3.5.4). In this case the {attribute declaration} of that attribute use is the ·context-determined declaration· for the attribute information item with respect to Schema-Validity Assessment (Attribute) (§3.2.4) and Assessment Outcome (Attribute) (§3.2.5).

2.2 **otherwise all** of the following are true:

2.2.1 There is an {attribute wildcard}.

2.2.2 And the attribute information item is ·valid· with respect to it as defined in Item Valid (Wildcard) (§3.10.4).

3 The {attribute declaration} of each attribute use in the {attribute uses} whose {required} is ***true*** matches one of the attribute information items in the element information item's [attributes] as per clause 2.1 above.

4 As part of ensuring that an element has no more than one attribute of type ID *{a little bit of up front text about why are set of rules are being applied helps set the brain in the right gear!}*, let [Definition:] the **wild IDs** be the set of all attribute information items to which clause 2.2 applied and whose ·validation· resulted in a ·context-determined declaration· of ***mustFind*** or no ·context-determined declaration· at all, and whose [local name] and [namespace name] resolve (as defined by QName resolution (Instance) (§3.16.4)) to an attribute declaration whose {type definition} is or is constructed from ID. Then **all** of the following are true:

4.1 There is no more than one item in ·wild IDs·.

4.2 If ·wild IDs· is non-empty, there are no attribute uses among the {attribute uses} whose {attribute declaration}.{type definition} is or is constructed from ID.

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

5 The element information item is ·valid· with respect to each of the {assertions} as per Assertion Satisfied (§3.12.4).

Note: When an {attribute wildcard} is present, this does *not* introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the {attribute uses} whose name and target namespace match are ·assessed·. In such cases the attribute use *always* takes precedence, and the ·assessment· of such items stands or falls entirely on the basis of the attribute use and its {attribute declaration}. This follows from the details of clause 2.

3.4.5 Complex Type Definition Information Set Contributions

Schema Information Set Contribution: Attribute Default Value
For each attribute use in the {attribute uses} whose {required} is ***false*** and whose {value constraint} is ~~not absent~~ present *{not absent feels a bit like not not there, i.e. a double negative. The addition of a feel additional terms, which can be defined in terms of those terms already defined, e.g. present = not absent, would help the clarity.}* but whose {attribute declaration} does not match one of the attribute information items in the element information item's [attributes] as per clause 2.1 of Element Locally Valid (Complex Type) (§3.4.4) above, the post-schema-validation infoset has an attribute information item whose properties are as below added to the [attributes] of the element information item.
Issue (RQ-22i): Issue 2852 (RQ-22 add normalized default)

Constructed default attribute information items in the PSVI did not have a [normalized value] property, only a [schema normalized value], making them incompatible with ordinary attribute infoitems. On balance, it seems sensible to correct this.

Resolution:

Add a [normalized value] property to the constructed attribute infoitem which arises when a default value is applied.

[local name]
The {attribute declaration}'s {name}.
[namespace name]
The {attribute declaration}'s {target namespace}.
[schema normalized value]
The effective value constraint's {lexical form}.
[schema default]
The effective value constraint's {lexical form}.
[validation context]
The nearest ancestor element information item with a [schema information] property.
[validity]
valid.
[validation attempted]
full.
[schema specified]
schema.
The added items MUST also either have [type definition] (and [member type definition] if appropriate) properties, or their lighter-weight alternatives, as specified in Attribute Validated by Type (§3.2.5).

3.4.6 Constraints on Complex Type Definition Schema Components

All complex type definitions (see Complex Type Definitions (§3.4)) MUST satisfy the following constraints.

Schema Component Constraint: Complex Type Definition Properties Correct

All of the following MUST be true:

1 The values of the properties of a complex type definition are as described in the property tableau in The Complex Type Definition Schema Component (§3.4.1), modulo the impact of Missing Sub-components (§5.3).

2 If the {base type definition} is a simple type definition, the {derivation method} is ***extension***.

3 There are no circular definitions, except for that of ***rootType***. That is, it is possible to reach the ||definition of ***rootType***|| by repeatedly following the {base type definition}.

4 No two distinct attribute declarations in the {attribute uses} have identical {name}s and {target namespace}s.

5 No two distinct attribute declarations in the {attribute uses} have {type definition}s which are or are constructed from ID.

Schema Component Constraint: Derivation Valid (Extension)

If the {derivation method} is ***extension***, then the appropriate **case** among the following MUST be true: *{avoid as many "it" and "it's" as possible}*

1 **If** the {base type definition} is a complex type definition, **then all** of the following are true:

1.1 The {base type definition}.{final} does not contain ***extension***.

1.2 And {base type definition}.{attribute uses} is a subset of the {attribute uses} of the complex type definition itself. That is, for every attribute use in {base type definition}.{attribute uses}, there is an attribute use in the {attribute uses} of the complex type definition itself whose {attribute declaration} has the same {name}, {target namespace} and {type definition} as its attribute declaration.

1.3 And if {base type definition} has an {attribute wildcard}, then the complex type definition also has one, and {base type definition}.{attribute wildcard}.{namespace constraint} is a subset of the complex type definition's {attribute wildcard}.{namespace constraint}, as defined by Wildcard Subset (§3.10.6).

1.4 And **One** of the following is true:

1.4.1 {content type}.{variety} and {base type definition}.{content type}.{variety} are both ***simple*** and{content type}.{simple type definition} is the same as {base type definition}.{content type}.{simple type definition}.

1.4.2 Or both {content type}.{variety} and {base type definition}.{content type}.{variety} are ***empty***.

1.4.3 Or **All** of the following are true:

1.4.3.1 {content type}.{variety} is ***element-only*** or ***mixed***.

1.4.3.2 And **One** of the following is true:

1.4.3.2.1 The {base type definition}.{content type}.{variety} is ***empty***.

1.4.3.2.2 Or **All** of the following are true:

1.4.3.2.2.1 Both {content type}.{variety} and {base type definition}.{content type}.{variety} are ***mixed*** or both are ***element-only***.

1.4.3.2.2.2 And {content type}.{particle} is a :valid extension: of {base type definition}.{content type}.{particle}, as defined in Particle Valid (Extension) (§3.9.6).

1.5 And It is in principle possible to derive the complex type definition in two steps, the first an extension and the second a restriction (possibly vacuous), from that type definition among its ancestors whose {base type definition} is the :ur-type definition:.

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

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

1.6 And The {base type definition}.{assertions} is a prefix of the {assertions} of the complex type definition itself.

2 Or **If** the {base type definition} is a simple type definition, **then all** of the following are true:

2.1 The {content type}.{variety} is ***simple*** and {content type}.{simple type definition} is the same simple type definition as {base type definition}.{content type}.{simple type definition}.

2.2 And {base type definition}.{final} does not contain ***extension***.

[Definition:] A complex type **T** is a **valid extension** of its {base type definition} if and only if **T** has a {derivation method} of ***extension*** and satisfies the constraint Derivation Valid (Extension) (§3.4.6).

Schema Component Constraint: Derivation Valid (Restriction, Complex)

If the {derivation method} is ***restriction*** **all** of the following MUST be true:

1 The {base type definition} is a complex type definition and {base type definition}.{final} does not contain ***restriction***.

{the spec has a number of cases deleted here. Hence the job in numbers.}

5 And **One** of the following is true:

5.1 The {base type definition} is the :ur-type definition:.

5.2 Or **All** of the following are true:

5.2.1 The {content type} of the complex type definition in which {content type}.{variety} is ***simple***

5.2.2 And **One** of the following is true:

5.2.2.1 {base type definition}.{content type}.{simple type definition} is a simple type definition from which {content type}.{simple type definition} is validly derived given the empty set as defined in Type Derivation OK (Simple) (§3.15.6).

5.2.2.2 Or {base type definition}.{content type}.{variety} is ***mixed*** and {base type definition}.{content type}.{particle} is ·emptiable· as defined in Particle Emptiable (§3.9.6).

5.3 Or **All** of the following are true:

5.3.1 {content type}.{variety} is ***empty***

5.3.2 And **One** of the following is true:

5.3.2.1 {base type definition}.{content type}.{variety} is ***empty***.

5.3.2.2 Or {base type definition}.{content type}.{variety} is ***elementOnly*** or ***mixed*** and {base type definition}.{content type}.{particle} is ·emptiable· as defined in Particle Emptiable (§3.9.6).

5.4 Or **All** of the following are true:

5.4.1 **One** of the following is true:

5.4.1.1 {content type}.{variety} is ***element-only*** and {base type definition}.{content type}.{variety} is not ***simple***.

5.4.1.2 Or {content type}.{variety} and {base type definition}.{content type}.{variety} are ***mixed***.

5.4.2 And {content type} ·restricts· {base type definition}.{content type} as defined in Content type restricts (§3.4.6).

6

||

The complex type definition ·restricts· the {base type definition} as defined in Complex type definition actually restricts (§3.4.6).

||

7 The {base type definition}.{assertions} is a prefix of the {assertions} of the complex type definition itself.

[Definition:] A complex type definition with {derivation method} ***restriction*** is a **valid restriction** of its {base type definition} if and only if the constraint Derivation Valid (Restriction, Complex) (§3.4.6) is satisfied.

{I decided to skip the material that follows this point.}

3.4.7 Built-in Complex Type ||Definitions||

||

There is a Complex Type Definition corresponding to the root of the type hierarchy present in every schema by definition:

||

||

Complex Type Definition of the root of the type hierarchy (the ur-type)

Property

Value
{name}
rootType
{target namespace}
http://www.w3.org/2001/XMLSchema
{base type definition}
Itself
{derivation method}
restriction
{content type}
A Content Type with the following properties:
{content type}.{variety}
mixed
{content type}.{simple type definition}
:absent:
{content type}.{particle}
a Particle with the properties shown below in Outer particle for rootType (§3.4.7).
{attribute uses}
The empty set
{attribute wildcard}
a wildcard with the following properties:
{attribute wildcard}.{namespace constraint}
A Namespace Constraint with the following properties:
{attribute wildcard}.{namespace constraint}.{variety}
any
{attribute wildcard}.{namespace constraint}.{namespaces}
The empty set
{attribute wildcard}.{namespace constraint}.{disallowed names}
The empty set
{attribute wildcard}.{process contents}
skip
{final}
The empty set
{context}
:absent:
{prohibited substitutions}
The empty set
{assertions}
The empty sequence
{abstract}
false
||
||

The outer particle of ***rootType*** contains a simple sequence:

||
||
Outer particle for rootType
Property
Value
{min occurs}
1
{max occurs}
1
{term}
a model group with the following properties:
{term}.{compositor}
sequence
{term}.{particles}
a list containing one particle with the properties shown below in Inner particle for rootType (§3.4.7).
||
||

The inner particle of ***rootType*** contains a ***skip*** wildcard:

||
||
Inner particle for rootType
Property
Value
{min occurs}
0
{max occurs}
unbounded
{term}
a wildcard with the following properties:
Property
Value
{namespace constraint}
A Namespace Constraint contributing the following properties:
{namespace constraint}.{variety}
any
{namespace constraint}.{namespaces}
The empty set
{namespace constraint}.{disallowed names}
The empty set
{process contents}
skip
||
||

The `mixed` content specification together with the ***skip*** wildcard and attribute specification produce the defining property for the root of the type hierarchy, namely that *every* type definition is (eventually) a restriction of it: its permissions and requirements are the least restrictive possible.

||

There is ||also ||a complex type definition ||for :anyType:|| present in every schema by definition. It has the following properties:

||

Complex Type Definition of anyType

Property
Value
{name}
anyType
{target namespace}
http://www.w3.org/2001/XMLSchema
{base type definition}
The built-in :root type definition:
{derivation method}
restriction
{content type}
A Content Type as follows:
{content type}.{variety}
mixed
{content type}.{particle}
a Particle with the properties shown below in Outer Particle for Content Type of anyType (§3.4.7).
{content type}.{simple type definition}
:absent:
{attribute uses}
The empty set
{attribute wildcard}
a wildcard with the following properties::
{attribute wildcard}.{namespace constraint}
A Namespace Constraint with the following properties:
{attribute wildcard}.{namespace constraint}{variety}
any
{attribute wildcard}.{namespace constraint}{namespaces}
The empty set
{attribute wildcard}.{namespace constraint}{disallowed names}
The empty set
{attribute wildcard}.{process contents}
lax
{final}

The empty set
{context}
:absent:
{prohibited substitutions}
The empty set
{assertions}
The empty sequence
{abstract}
false
||

The outer particle of :anyType: contains a sequence with a single term:

Outer Particle for Content Type of anyType

{min occurs}
1
{max occurs}
1
{term}
a model group with the following properties:

Property
Value

{compositor}
sequence
{particles}
a list containing one particle with the properties shown below in Inner Particle for Content Type of anyType (§3.4.7).

The inner particle of :anyType: contains a wildcard which matches any element:

Inner Particle for Content Type of anyType

Property
Value

{min occurs}
0
{max occurs}
unbounded
{term}
a wildcard with the following properties:

Property
Value

{namespace constraint}
A Namespace Constraint with the following properties:
{namespace constraint}.{variety}
any
{namespace constraint}.{namespaces}

The empty set
{namespace constraint}.{disallowed names}
The empty set
{process contents}
lax

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.