

# E

## XML Schema Element and Attribute Reference

This appendix provides a full listing of all elements within the XML Schema Structures Recommendation (found at <http://www.w3.org/TR/xmlschema-1/>). The elements appear in alphabetical order. Each element is described with examples and a table detailing all the attributes used in the element. When attributes are required, it is noted in the attribute listings.

The end of this appendix presents a table of the attributes in the XML Schema Instance namespace that can be used in instance documents.

### ***all***

The `<all>` element is used within content model declarations. It indicates that all elements declared within it may appear in the instance document in any order and may appear at most once. The `<all>` element is used within a `<complexType>` or `<group>` element. It can contain `<element>` or `<annotation>` elements. Note that when using `minOccurs` and `maxOccurs` on `<element>` declarations within an `<all>` element, you are restricted to using a `maxOccurs` of 1 or 0. You can make an element optional by setting the `minOccurs` to 0. For more information, see §3.8.2 of the Recommendation.

### ***Example***

```
<xs:element name="Rucksack">
  <xs:complexType>
    <xs:all>
      <xs:element name="Sunglasses" type="xs:string" maxOccurs="1" />
      <xs:element name="Sweater" type="xs:string" maxOccurs="1" />
      <xs:element name="Book" type="xs:string" />
      <xs:element name="Lunchbox" type="xs:string" />
      <xs:element name="Flask" type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

## Appendix E: XML Schema Element and Attribute Reference

---

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
maxOccurs	1	The maximum number of times the <all> model group can occur
minOccurs	0 or 1	The minimum number of times the <all> model group can occur

### annotation

The <annotation> element is used to provide additional data for XML Schema declarations. It may contain the <appinfo> and <documentation> elements, which are used to contain instructions for the XML Schema processing application or for additional documentation. It is contained by most elements (excluding itself); specific cases are detailed in the following examples. For more information, see §3.13.2 of the Recommendation.

#### Example

This example uses <annotation> with <documentation>:

```
<xs:element name="Person">
  <xs:annotation>
    <xs:documentation>
      Used to contain personal information. Note that the last name
      is mandatory, while the first name is optional.
    </xs:documentation>
  </xs:annotation>
  <!-- definition of Person element goes here -->
</xs:element>
```

This example uses <annotation> with <appinfo>:

```
<xs:element name="Person" type="PersonType">
  <xs:annotation>
    <xs:appinfo>
      <sch:pattern name="Top Level Person elements">
        <sch:rule context="/*">
          <sch:assert test="self::Person">
            The root element must be a "Person"
          </sch:assert>
        </sch:rule>
      </sch:pattern>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this second example, the <annotation> element is used to contain a Schematron schema inside the <appinfo> element.

## any

The `<any>` element is used within content model declarations. It is a wildcard element that acts as a placeholder for any element in a model group. Using the `<any>` declaration it is possible to specify from which namespaces allowable elements may come. This is useful, for instance, if unspecified XHTML or MathML content might be included within the instance document. It may contain an `<annotation>` element, and can be contained by `<choice>` or `<sequence>` elements. For more information, see §3.10.2 of the Recommendation.

### Example

```
<xs:element name="XHTMLSection">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="0" maxOccurs="unbounded"
        processContents="lax" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Here, an `XHTMLSection` element in an instance document can contain any well-formed markup that is valid in the XHTML namespace.

### Attributes

Attribute	Value Space	Description
<code>id</code>	ID	Gives a unique identifier to the element
<code>maxOccurs</code>	nonNegativeInteger or unbounded	The maximum number of times the model group can occur
<code>minOccurs</code>	nonNegative Integer	The minimum number of times the model group can occur
<code>namespace</code>	##any   ##other    List of (anyURI   ##targetNamespace   ##local)	##any means that the content can be from any namespace. ##other refers to any namespace other than the target namespace of the schema.  The value can also be a list of actual namespaces such as <code>http://www.example.com/name</code> . Additionally the list can include the value <code>##targetNamespace</code> to allow elements in the target namespace of the schema, and <code>##local</code> to allow elements in no namespace. The default is <code>##any</code> .
<code>process Contents</code>	skip   lax   strict	If <code>lax</code> , then validation is performed only when an associated schema can be found for the wildcard elements. If <code>skip</code> , then no validation occurs. If <code>strict</code> , then validation is enforced, and the validator needs access to the declarations for the elements used or a validity error will be raised. The default is <code>skip</code> .

### ***anyAttribute***

The `<anyAttribute>` element is used within content model declarations. It acts as a placeholder for any attribute within an element declaration. It allows any unspecified attributes to be present. These can be validated against a specific namespace. For example, XML Schema documents allow elements to have any attributes as long as they're not in the XML Schema namespace and are qualified with a prefix for another namespace. You might encounter a situation where you need to allow the use of any XLink attribute within a specific element. The `<anyAttribute>` can be contained by `<attributeGroup>`, `<complexType>`, `<extension>`, or `<restriction>` elements; and like most elements it can contain an annotation. For more information, see §3.4.2 of the Recommendation.

#### **Example**

```
<xs:element name="Description">
  <xs:complexType>
    <!-- content definition goes here-->
    <xs:anyAttribute namespace="http://www.w3.org/1999/xlink" />
  </xs:complexType>
</xs:element>
```

Here, a `Description` element in an instance document can contain any attribute that is valid in the XLink namespace.

#### **Attributes**

Attribute	Value Space	Description
<code>id</code>	ID	Gives a unique identifier to the element
<code>namespace</code>	<code>##any</code>   <code>##other</code>   List of (anyURI   <code>##targetNamespace</code>   <code>##local</code> ) "	<code>##any</code> means that the content can be of any namespace. <code>##other</code> refers to any namespace other than the target namespace of the schema. (The attributes must be namespace qualified.)  The value can also be a list of actual namespaces such as <code>http://www.example.com/name</code> . Additionally the list can include the value <code>##targetNamespace</code> to allow elements in the target namespace of the schema, and <code>##local</code> to allow elements in no namespace. The default is <code>##any</code> .
<code>processContents</code>	<code>skip</code>   <code>lax</code>   <code>strict</code>	If <code>lax</code> , then validation is performed only if an associated schema can be found for the wildcard attributes. If <code>skip</code> , then no validation occurs. If <code>strict</code> , then validation is enforced, and the validator needs access to the declarations for the attributes used or a validity error will be raised. The default is <code>skip</code> .

## appinfo

The `<appinfo>` element is used within `<annotation>` declarations. It allows information to be supplied to an application reading the schema, and may contain unique identifiers or additional tags to help an application perform further processing on the schema. Although the XML Schema Recommendation does not specify allowable uses for the `<appinfo>` element, many XML Schema designers use it to combine Schematron validation with XML Schema validation. For more information on combining XML Schema and Schematron validation, see [http://www.topologi.com/public/Schtrn\\_XSD/Paper.html](http://www.topologi.com/public/Schtrn_XSD/Paper.html). Multiple `<appinfo>` elements may appear within a single `<annotation>` declaration. For more information, see §3.13.2 of the Recommendation.

### Example

```
<xs:element name="Person" type="PersonType">
  <xs:annotation>
    <xs:appinfo>
      <sch:pattern name="Top Level Person elements">
        <sch:rule context="/*">
          <sch:assert test="self::Person">
            The root element must be a "Person"
          </sch:assert>
        </sch:rule>
      </sch:pattern>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

### Attributes

Attribute	Value Space	Description
source	anyURI	Specifies a URI where the parser can acquire the required <code>&lt;appinfo&gt;</code> content. If the <code>source</code> attribute is not included, then the parser will check the contents of the <code>&lt;appinfo&gt;</code> element.

## attribute

The `<attribute>` element is used to declare allowable attributes within elements. It is usually found within an `<attributeGroup>` or a `<complexType>` element and defines the attributes for that particular content model. It can also be used in an `<extension>` or `<restriction>` element when deriving a new type. Attribute declarations may appear in the root `<schema>` element to create global attribute definitions that can be referenced from other declarations. The `<attribute>` element may contain an `<annotation>` element. It may also contain an anonymous `<simpleType>` declaration if no `type` attribute is specified. For more information, see §3.2 of the Recommendation.

## Appendix E: XML Schema Element and Attribute Reference

---

### Example

```
<xs:attribute name="Amount">
  <xs:simpleType name="positiveDecimalN.2" >
    <xs:restriction base="xs:decimal" >
      <xs:minInclusive value="0" />
      <xs:fractionDigits value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<xs:element name="Payment">
  <xs:complexType >
    <xs:attribute ref="Amount" />
    <xs:attribute name="Currency" type="xs:string" default="US"
      use="optional" />
  </xs:complexType>
</xs:element>
```

### Attributes

Attribute	Value Space	Description
default	string	A string containing a default value for the attribute that is used if the attribute is not specified in the instance document
fixed	string	If present, the value of the attribute in an instance document must always match the value specified by <code>fixed</code> .
form	qualified unqualified	If <code>qualified</code> , the attribute must be namespace qualified in the instance document. Note that if the <code>form</code> attribute is present on the attribute element, then it overrides <code>attributeFormDefault</code> on the schema element. All global attribute declarations must be qualified regardless of the value of the <code>form</code> attribute or <code>attributeFormDefault</code> attribute. For an attribute to be qualified in an instance document, it must have a prefix associated with the namespace; default namespace declarations do not apply to attributes.
id	ID	Gives a unique identifier to the element
name	NCName	The name of the attribute conforming to the XML NCName data type
ref	QName	Refers a previously defined global attribute by name. The <code>ref</code> attribute cannot be used in global <code>&lt;attribute&gt;</code> declarations
type	QName	The data type of the attribute
use	optional   prohibited   required	If <code>optional</code> , the attribute may be omitted in the instance document. If <code>required</code> , it must be included. If <code>prohibited</code> , it cannot be included. The default is <code>optional</code> .

## **attributeGroup**

The `<attributeGroup>` element is used to declare a group of attributes or to refer to an existing global `<attributeGroup>` declaration. This is useful when more than one element contains the same group of attributes. It may contain `<annotation>`, `<attribute>`, `<attributeGroup>`, and `<anyAttribute>` declarations. You can create a global declaration for a group of attributes by declaring the `<attributeGroup>` element as a direct child of the `<schema>` element. Attribute group definitions can be nested, so an `<attributeGroup>` can contain or be contained by another `<attributeGroup>`. It can also be used as a reference from within a `<complexType>`, `<redefine>`, `<extension>`, or `<restriction>` declaration. For more information, see §3.6.2 of the Recommendation.

### **Example**

```
<xs:attributeGroup name="PhysicalDescriptionAttrGroup">
  <xs:attribute name="weight" type="xs:decimal" use="optional" />
  <xs:attribute name="height" type="xs:decimal" use="optional" />
</xs:attributeGroup>

<xs:element name="Person">
  <xs:complexType>
    <xs:sequence>
      <!-- element content here -->
    </xs:sequence>
    <xs:attributeGroup ref="PhysicalDescriptionAttrGroup" />
  </xs:complexType>
</xs:element>
```

### **Attributes**

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
name	NCName	The name of this attribute group
ref	QName	Refers to a previously defined global attribute group; used within a <code>&lt;complexType&gt;</code> definition to include a group of attributes. The <code>ref</code> attribute cannot be used in global <code>&lt;attributeGroup&gt;</code> declarations.

## **choice**

The `<choice>` element is used within content model declarations. It is used to indicate that only one of its contained declarations can be used within the content model in the instance document. It may contain `<annotation>` and `<element>` declarations. In addition, because you can nest content models, it may contain `<choice>`, `<sequence>`, `<group>`, and `<any>` elements. Similarly, it can be contained by `<choice>`, `<group>`, `<sequence>`, or `<complexType>` elements. For more information, see §3.8.2 of the Recommendation.

### Example

```
<xs:element name="IceCream">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="Strawberry" type="xs:string" />
        <xs:element name="Chocolate" type="xs:string" />
      </xs:choice>
      <xs:choice>
        <xs:element name="Cone" type="xs:string" />
        <xs:element name="Tub" type="xs:string" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
maxOccurs	nonNegative Integer or unbounded	The maximum number of times the model group can occur
minOccurs	nonNegative Integer	The minimum number of times the model group can occur

## complexContent

The `<complexContent>` element is used when descending new complex types using extension or restriction. It indicates that the resulting content model can have attributes and can contain element content or mixed content or even be empty. This element is used inside a `<complexType>` declaration and can contain an `<annotation>`, `<restriction>`, or `<extension>` element. For more information, see §3.4.2 of the Recommendation.

### Example

```
<xs:complexType name="CAN_Address">
  <xs:complexContent>
    <xs:extension base="Address">
      <xs:sequence>
        <xs:element name="Province" type="xs:string" />
        <xs:element name="PostalCode" type="CAN_PostalCode" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
mixed	boolean	If <code>true</code> , the element can contain text and element content. The default is <code>false</code> .

## complexType

The `<complexType>` element is used to specify the allowable type of content for elements. Complex type definitions are the key to the creation of complex structures and content models in XML Schemas. They should be used when an element will contain anything that is more complex than simple character data, such as attributes and child elements. A `<complexType>` can be declared globally (for example, as a direct child of the `<schema>` element) or locally (for example, as a direct child of an `<element>` declaration). They can also be used from within a `<redefine>` element. A `<complexType>` may contain an optional `<annotation>` element. It may be derived from another type, in which case it must contain a `<simpleContent>` or `<complexContent>` element. Alternatively, you can specify the allowable content model directly using `<group>`, `<all>`, `<choice>`, or `<sequence>` elements, followed by attribute declarations using `<attribute>`, `<attributeGroup>`, or `<anyAttribute>` elements. For more information, see §3.4.2 of the Recommendation.

### Example

```
<xs:element name="ResearchPaper">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Hypothesis" type="xs:string" />
      <xs:element name="Conclusion" type="ConclusionType" />
    </xs:sequence>
    <xs:attribute name="paperID" type="xs:integer" />
  </xs:complexType>
</xs:element>

<xs:complexType name="ConclusionType" block="#all">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="accepted" type="xs:boolean" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## Appendix E: XML Schema Element and Attribute Reference

---

### Attributes

Attribute	Value Space	Description
abstract	boolean	This specifies whether the complex type can be used to validate an element. If <code>abstract</code> is <code>true</code> , then it can't; you have to derive other types from it for use in an instance document. Note that this behavior is distinct from using the <code>abstract</code> attribute on an <code>element</code> declaration (for more information, refer to "element," later in this appendix). The default is <code>false</code> .
block	#all   List of (extension   restriction)	Enables the schema author to prevent derived types from being used in the instance document in place of this type. The values <code>extension</code> and <code>restriction</code> prevent the use of types derived by extension and restriction, respectively, and <code>#all</code> prevents the use of any derived type.
final	#all   List of (extension   restriction)	This attribute restricts the derivation of a new data type by <code>extension</code> or <code>restriction</code> within the schema. The values <code>extension</code> and <code>restriction</code> prevent the creation of types derived by extension and restriction, respectively, and <code>#all</code> prevents the creation of any derived type.
id	ID	Gives a unique identifier to the type
Mixed	boolean	Specifies whether the content of this data type is mixed
Name	NCName	The name of the complex data type being declared

### documentation

The `<documentation>` element is used within `<annotation>` declarations. It provides a consistent location for comments about the declarations in your XML Schema. The `<documentation>` element allows any content (such as well-formed XHTML), and external references can be made using the `source` attribute. Though the XML Schema Recommendation does not outline specific uses for the `<documentation>` element, many XML Schema designers use it to produce automatically generated help files for their XML Schemas. Multiple `<documentation>` elements may appear within a single `<annotation>` declaration. For more information, see §3.13.2 of the Recommendation.

### Example

```
<xs:element name="Person">
  <xs:annotation>
    <xs:documentation>
      Used to contain personal information. Note that the last name
      is mandatory, while the first name is optional.
    </xs:documentation>
  </xs:annotation>
  <!-- definition of Person element goes here -->
</xs:element>
```

## Attributes

Attribute	Value Space	Description
Source	anyURI	Specifies the URI where the content of this element may be found. You don't need this attribute if the content is specified within the documentation tag, as in the previous example.
xml:lang	language	Specifies the language, using a code defined by RFC 3066. Most languages can be identified by a simple two-letter code.

## element

The `<element>` declaration is possibly the most important schema namespace element because it is used to declare the elements that can occur in the instance document. It may contain a `<simpleType>` or a `<complexType>`, creating a local type for the allowable content. Alternatively, the type of content may be specified using the `type` attribute. The `<element>` declaration may also contain `<unique>`, `<key>`, or `<keyref>` elements to define identity constraints. As with most elements, it may also contain an `<annotation>`. Elements are declared within model groups using `<all>`, `<choice>`, or `<sequence>`, or can be declared globally as children of the `<schema>` element. For more information, see §3.3.2 of the Recommendation.

## Example

```
<xs:element name="Customer">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string" />
      <xs:element name="MiddleInitial" type="xs:string" />
      <xs:element name="LastName" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="customerID" type="xs:string" />
  </xs:complexType>
</xs:element>
```

## Attributes

Attribute	Value Space	Description
abstract	boolean	Specifies that the element is abstract and cannot appear in the instance document, but must be substituted with another element. The default is <code>false</code> .
block	#all   List of (substitution   extension   restriction)	Prevents derived types from being used in place of this element in the instance document (which can be done with the <code>xsi:type</code> attribute), and/or substituting another element in its place. The values <code>extension</code> and <code>restriction</code> prevent the use of types derived by extension and restriction, respectively, and <code>#all</code> prevents the use of any derived type.

*Table continued on following page*

## Appendix E: XML Schema Element and Attribute Reference

Attribute	Value Space	Description
default	string	This attribute enables you to specify a default value for the element, which is used when the element appears in the instance document but is empty.
final	#all   List of (extension   restriction)	Prevents the element from being nominated as the head element in a substitution group, which has members derived by extension and/or restriction as appropriate.
fixed	string	If present, the value of the element in the instance document must always match the specified fixed value.
form	qualified   unqualified	If qualified, the element must be namespace qualified in the instance document. The value of this attribute overrides whatever is specified by the <code>elementFormDefault</code> on the schema element. All global element declarations must be qualified regardless of the value of the <code>form</code> attribute or <code>elementFormDefault</code> attribute.
id	ID	Gives a unique identifier to the type
maxOccurs	nonNegative Integer   unbounded	The maximum number of times the element can occur. Global element declarations can't use the <code>maxOccurs</code> attribute.
minOccurs	nonNegative Integer	The minimum number of times the element can occur. Global element declarations cannot use the <code>minOccurs</code> attribute.
name	NCName	The name of the element
nillable	boolean	If true, the element may have a nil value specified with <code>xsi:nil</code> in the instance document. The default is false.
ref	QName	Enables you to reference a globally defined element using the value of that element's <code>name</code> attribute. The <code>ref</code> attribute can't be used in global <code>&lt;element&gt;</code> declarations.
substitution Group	QName	The element becomes a member of the substitution group specified by this attribute. Wherever the head element of the substitution group is used in a model group, you can substitute this element in its place.
type	QName	The type of content of this element, which could be simple or complex. If the element contains a <code>&lt;simpleType&gt;</code> or <code>&lt;complexType&gt;</code> element, the <code>type</code> attribute must not be used.

## extension

The `<extension>` element is used when descending new complex types. Using this declaration, you can extend a base type by adding additional element or attribute declarations. When adding element content to a type, the extension element may contain `<group>`, `<choice>` or `<sequence>` elements. When adding attributes, it will contain one or more `<attribute>`, `<attributeGroup>`, or `<anyAttribute>` declarations. Note that when an `<extension>` element is contained inside a `<complexContent>` declaration it can introduce new elements and/or attributes, whereas when it is inside a `<simpleContent>` declaration it can be used only to add attributes to a type. For more information, see §3.4.2 of the Recommendation.

### Example

Extending a complex type:

```
<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="country" type="xs:string"/>
    <xs:element name="address" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CANAddress">
  <xs:complexContent>
    <xs:extension base="Address" >
      <xs:sequence>
        <xs:element name="province" type="xs:string"/>
        <xs:element name="postalcode" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="USAddress">
  <xs:complexContent>
    <xs:extension base="Address" >
      <xs:sequence>
        <xs:element name="state" type="xs:string"/>
        <xs:element name="zipcode" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Extending a simple type to produce a complex type with simple content:

```
<xs:complexType name="ConclusionType" block="#all">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="accepted" type="xs:boolean" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## Appendix E: XML Schema Element and Attribute Reference

---

### Attributes

Attribute	Value Space	Description
base (required)	QName	Specifies the base internal or derived data type that will be extended
id	ID	Gives a unique identifier to the element

### field

The `<field>` element is used when creating identity constraints, such as `<key>`, `<keyref>`, and `<unique>` declarations. An *identity constraint* allows you to both specify that certain nodes in the document are unique and create relationships between multiple nodes in a document using the node's unique or shared identity. By default, XML allows you to use the built in ID and IDREF identity constraints. With XML Schema you can use most datatypes as part of an identity constraint, and apply them to attributes or elements. When creating identity constraints, you must specify a context, or scope, for the constraint using a `<selector>` declaration, and the specific node that is constrained using a `<field>` declaration. It may contain an `<annotation>` element. For a complete example, please see key. For more information, see §3.11.2 of the Recommendation.

### Example

```
<xs:element name="Employees">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="example:Employee" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="employeeIdentificationNumber">
    <xs:selector xpath="example:Employee" />
    <xs:field xpath="@employeeID" />
  </xs:unique>
</xs:element>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
xpath (required)	XPath	Used to select the element context affected by the identity constraint. The path is relative to the current element declaration.

### group

The `<group>` element is used to declare a group of elements or content model declarations or to refer to an existing global `<group>` declaration. This is useful when more than one element contains the same content model. When the `<group>` element is a direct child of the `<schema>` element, it must be used as

## Appendix E: XML Schema Element and Attribute Reference

a global declaration for a content model group and it may contain a `<sequence>`, `<choice>`, or `<all>` declaration. It can also be used as a reference from within a `<complexType>`, `<redefine>`, `<extension>`, or `<restriction>` declaration. Because content models can be nested, the `<group>` element can also be referenced within a `<sequence>` or `<choice>` declaration. For more information, see §3.7.2 of the Recommendation.

### Example

```
<xs:element name="Customer">
  <xs:complexType>
    <xs:group ref="FirstOrLastNameGroup" />
  </xs:complexType>
</xs:element>

<xs:group name="FirstOrLastNameGroup">
  <xs:choice>
    <xs:element name="FirstName" type="xs:string" />
    <xs:element name="LastName" type="xs:string" />
  </xs:choice>
</xs:group>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
maxOccurs	nonNegative Integer   unbounded	The maximum number of times the group can occur
minOccurs	nonNegative Integer	The minimum number of times the group can occur
name	NCName	Defines the name of a global model group. If you are creating a global model group, the <code>ref</code> , <code>minOccurs</code> , and <code>maxOccurs</code> attributes are not permitted.
ref	QName	Refers to a previously defined global group. When using this attribute, you cannot include a <code>name</code> attribute, but you can set occurrence constraints with <code>minOccurs</code> and/or <code>maxOccurs</code> . The <code>ref</code> attribute cannot be used in global <code>&lt;group&gt;</code> declarations.

### import

The `<import>` declaration is used to combine multiple XML Schemas. It enables you to import the declarations from an XML Schema for another namespace. If you are trying to combine XML Schemas that utilize the same namespace or have no namespace, you should instead use the `<include>` declaration. The `<import>` element should be declared as a child of the root `<schema>` element, and has an optional `<annotation>`. An XML Schema may contain multiple `<import>` declarations. For more information, see §4.2.3 of the Recommendation.

## Appendix E: XML Schema Element and Attribute Reference

---

### Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/Order"
  xmlns="http://www.example.com/Order"
  xmlns:products="http://www.example.com/Products"
  xmlns:types="http://www.example.com/Types"
  elementFormDefault="qualified">

  <xs:import schemaLocation="Products.xsd"
    namespace="http://www.example.com/Products" />
  <xs:import schemaLocation="TypeLib.xsd"
    namespace="http://www.example.com/Types" />

  <!-- rest of schema definition here -->
</xs:schema>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
namespace	anyURI	The namespace of the imported declarations
schemaLocation	anyURI	The location of the schema to import

## include

The `<include>` declaration is used to combine multiple XML Schemas that have the same target namespace, or no target namespace. If you include an XML Schema with no target namespace, the declarations will be treated as if they were declared using the target namespace of the including XML Schema. If you are trying to combine XML Schemas that utilize different namespaces, you should instead use the `<import>` declaration. The `<include>` element should be declared as a child of the root `<schema>` element, and has an optional `<annotation>`. An XML Schema may contain multiple `<include>` declarations. Reusing existing definitions is good practice — it saves you time when creating the documents and increases your document's interoperability. Utilizing the `<include>` declaration is ideal in a team environment, when you need to develop and maintain distinct parts of a large schema. For more information, see §4.2.1 of the Recommendation.

### Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/ECommerce"
  xmlns="http://www.example.com/ECommerce"
  elementFormDefault="qualified">

  <xs:include schemaLocation="Products.xsd" />
  <xs:include schemaLocation="TypeLib.xsd" />

  <!-- rest of schema definition here -->
</xs:schema>
```

## Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
schemaLocation (required)	anyURI	The location of the schema to include

## key

The `<key>` declaration, along with the `<keyref>` declaration, enables you to define a relationship between two elements. The key/keyref mechanism functions similarly to database keys or to the ID/IDREF mechanism built into XML DTDs. For example, an element might contain a `<key>` that is unique within a specified context or scope. Another element can refer to the key using a `<keyref>` element. A `<key>` is always defined inside an `<element>` declaration. It contains a `<selector>` element that defines the context or scope of the key, and a `<field>` element that defines the specific key node. Like other elements, it can also contain an `<annotation>`. For more information, see §3.11.2 of the Recommendation.

## Example

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://example.com"
  xmlns:example="http://example.com"
  targetNamespace="http://example.com"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="Company">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Departments">
          <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
              <xs:element name="Department">
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:attribute name="building" type="xs:string"/>
                  <xs:attribute name="departmentID" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Employees">
          <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
              <xs:element name="Employee">
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:attribute name="position" type="xs:string"/>
                  <xs:attribute name="department" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Appendix E: XML Schema Element and Attribute Reference

```
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:key name="KeyDepartmentByID">
  <xs:selector xpath="example:Departments/example:Department" />
  <xs:field xpath="@departmentID" />
</xs:key>
<xs:keyref name="RefEmployeeToDepartment" refer="example:KeyDepartmentByID">
  <xs:selector xpath="example:Employees/example:Employee" />
  <xs:field xpath="@department" />
</xs:keyref>
</xs:element>
</xs:schema>
```

A corresponding instance document might be as follows:

```
<Company
  xmlns="http://example.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.com test2.xsd">
  <Departments>
    <Department name="Human Resources" building="Building 1"
      departmentID="hr_dept" />
    <Department name="Development" building="Building 2"
      departmentID="development_dept" />
    <Department name="Testing" building="Building 2" departmentID="testing_dept" />
  </Departments>
  <Employees>
    <Employee name="Oliver" position="Developer" department="development_dept" />
    <Employee name="Mwatha" position="Developer" department="development_dept" />
    <Employee name="Soyapi" position="Developer" department="development_dept" />
    <Employee name="Mike" position="Testing" department="testing_dept" />
  </Employees>
</Company>
```

It is important to recognize that you have to explicitly refer to each element's namespace in the selector and field `xpath` attributes. Even though there is a default namespace declaration in the document, it is not applied to the XPath statements in keys and key references. Therefore, you must use a namespace prefix (in this case `example`) to refer to the elements if there is a target namespace in your XML Schema. Though identity constraints are now widely supported, the quality of support varies. For example, some processors require that you use a namespace prefix in the `refer` attribute as well.

### Attributes

Attribute	Value Space	Description
<code>id</code>	ID	Gives a unique identifier to the element
<code>name</code> (required)	NCName	The name of the key used

## keyref

The `<keyref>` element is used to specify a reference to a `<key>` (see the previous discussion of `<key>`). Like the `<key>` element it may be used within an `<element>` declaration. It may contain an `<annotation>` element, and can define the context of the key reference by including a `<selector>` declaration and `<field>` declaration. For a complete example, please see `key`. For more information, see §3.11.2 of the Recommendation.

### Example

```
<xs:keyref name="RefEmployeeToDepartment" refer="example:KeyDepartmentByID">
  <xs:selector xpath="example:Employees/example:Employee" />
  <xs:field xpath="@department" />
</xs:keyref>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
name (required)	NCName	The name of the key reference
refer (required)	QName	The name of the key to which this key reference refers

## list

The `<list>` element is used to declare a specialized simple type, which is a sequence of whitespace-separated `<simpleType>` names. The `itemType` attribute defines the allowable type for each item contained in the list. Because you cannot create a list of lists, the `itemType` cannot refer to an existing list type. Moreover, because lists use whitespace to separate the values, item types that refer to any type that can contain whitespace can be problematic. For example, the XML Schema `string` value may contain spaces, such as "This is a string of text". When treated as a list, an XML Schema processor would see six separate values, not one value with five spaces. A `<list>` declaration must appear within a `<simpleType>` definition and can contain optional `<annotation>` and `<simpleType>` elements. For more information, see §3.14.2 of the Recommendation.

### Example

```
<xs:simpleType name="AgesList">
  <xs:list itemType="xs:integer" />
</xs:simpleType>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
itemType	QName	The base data type for each item in the list

### **notation**

A `<notation>` declaration is used to associate a particular type of file with the location of an application that can process it. Within XML Schemas, notations must be declared globally (the `<notation>` element must be a direct child of the `<schema>` element). A `<notation>` declaration has a global name that is specified using the `name` attribute. In addition to the name, the `<notation>` provides `public` and `system` attributes that can be used to specify the public identifier and system identifier, respectively. The public identifier is optional. In general, the `<notation>` declaration should be avoided because of compatibility issues and poor implementation support. For more information, see §3.12.2 of the Recommendation.

#### **Example**

```
<xs:notation name="jpeg" public="image/jpeg" system="JPEGViewer.exe" />
<xs:notation name="png" public="image/png" system="PNGViewer.exe" />

<xs:simpleType name="ImageTypeNotation" >
  <xs:restriction base="xs:NOTATION">
    <xs:enumeration value="jpeg"/>
    <xs:enumeration value="png"/>
  </xs:restriction>
</xs:simpleType>
```

#### **Attributes**

Attribute	Value Space	Description
<code>id</code>	ID	Gives a unique identifier to the element
<code>name</code> (required)	NCName	The name of the specified NOTATION data type
<code>public</code>	anyURI	Any URI; usually some relevant identifier, such as a Multipurpose Internet Mail Extension (MIME) type. MIME types are used to identify file types on the World Wide Web. The MIME types for XML include <code>text/xml</code> and <code>application/xml</code> .
<code>system</code>	anyURI	Any URI; usually some local processing application

### **redefine**

The `<redefine>` declaration is used to combine multiple XML Schemas. It enables you to modify complex types, simple types, model groups, or attribute groups as they are included from another external schema. The external schema must have no namespace or it must have the same target namespace as the schema where `<redefine>` is used. Within the `<redefine>` element, you must refer to an existing type and amend it as necessary using extension or restriction. A `<redefine>` declaration must appear within the root `<schema>` element, and may contain `<annotation>`, `<simpleType>`, `<complexType>`, `<group>`, or `<attributeGroup>` elements. Though this seems cumbersome, it can be very useful to override existing schema definitions using `<redefine>` declarations. Because this feature allows you to modify existing declarations without modifying the existing schemas it is very powerful. However, it is often not supported by tools that attempt to build programming language bindings from an XML Schema, so use it sparingly. For more information, see §4.2.2 of the Recommendation.

## Example

From one schema you have the following:

```
<xs:complexType name="NameType">
  <xs:sequence>
    <xs:element name="FirstName" type="xs:string" />
    <xs:element name="MiddleInitial" type="xs:string" minOccurs="0" />
    <xs:element name="LastName" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

You can redefine this in another schema like so:

```
<xs:redefine schemaLocation="firstSchema.xsd">
  <xs:complexType name="NameType">
    <xs:complexContent>
      <xs:restriction base="NameType">
        <xs:sequence>
          <xs:element name="FirstName" type="xs:string" />
          <xs:element name="LastName" type="xs:string" />
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
```

## Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
schemaLocation (required)	anyURI	Specifies the location of the schema

## restriction

Use the `<restriction>` element when descending new complex types. Using this declaration, you can restrict a base type and limit the allowable content within `<complexType>` or `<simpleType>` declarations. You might use restriction in three different situations: to restrict a simple type, to restrict a complex type using simple content, or to restrict a complex type using complex content.

When restricting `<complexType>` declarations you must start with a base type and create the derivation by removing elements or attributes. Instead of specifying which elements you want to remove, when creating your restricted `<complexType>` declarations you must *redeclare* all elements you want to keep. Because of this, restricting `<complexType>` declarations is far more difficult than extending them. By default, attributes are automatically included in the newly restricted type.

The rules for restricting a `<complexType>` are very involved. Instead of listing all the conditions and exceptions, we focus on two basic rules: First, you cannot introduce anything new when restricting a

## Appendix E: XML Schema Element and Attribute Reference

---

<complexType>. Essentially, this means that you can't add elements or attributes that don't exist in the base type. When modifying existing declarations you must also be careful that the modifications are permitted. Second, you cannot remove anything that must appear in the base type. For example, if your base type declares that an element has a `minOccurs` value of 1 (the default), it cannot be removed in your restriction. This rule was created so that applications designed to handle the base type can also handle the restricted type without raising an error.

The <restriction> element may appear inside <simpleType>, <simpleContent>, or <complexContent>. In the first two situations, the element may contain a <simpleType> element and one of the constraining facets: <minExclusive>, <maxExclusive>, <minInclusive>, <maxInclusive>, <totalDigits>, <fractionDigits>, <length>, <minLength>, <maxLength>, <enumeration>, <whiteSpace>, or <pattern>. When restricting a <complexType>, a <restriction> declaration may also contain <attribute>, <attributeGroup>, and <anyAttribute>. If the <restriction> declaration appears inside a <complexContent> element, it may also include <group>, <all>, <choice>, and <sequence> declarations. The <restriction> element also has an optional <annotation> element. For more information, see §3.4.2 and §3.14.2 of the Recommendation.

### Example

Here's how you can derive a simple type:

```
<xs:simpleType name="Char">
  <xs:restriction base="xs:string">
    <xs:length value="1" />
  </xs:restriction>
</xs:simpleType>
```

Here's the code for deriving a complex type with simple content:

```
<xs:complexType name="Person">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="age" type="xs:integer" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:complexType name="RestrictedPerson">
  <xs:simpleContent>
    <xs:restriction base="Person">
      <xs:attribute name="age">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1" />
            <xs:maxInclusive value="120" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Here's how to derive a complex type with complex content:

```
<xs:complexType name="ShortAddress">
  <xs:complexContent>
    <xs:restriction base="Address" >
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Street" type="xs:string" maxOccurs="2" />
        <xs:element name="City" type="xs:string" />
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
base (required)	QName	The base type from which the new type is derived

## schema

This `<schema>` element is the root element within an XML Schema. Details, such as target namespace and global defaults, are specified within the `<schema>` element. It may contain `<include>`, `<import>`, `<redefine>`, `<annotation>`, `<simpleType>`, `<complexType>`, `<group>`, `<attributeGroup>`, `<element>`, `<attribute>`, or `<notation>`. For more information, see §3.15.2 of the Recommendation.

### Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/"
  xmlns="http://www.example.com/"
  elementFormDefault="qualified">
  <!--rest of content goes here-->
</xs:schema>
```

## Appendix E: XML Schema Element and Attribute Reference

---

### Attributes

Attribute	Value Space	Description
attributeFormDefault	qualified   unqualified	Enables you to specify a default for attribute qualification in the instance document. If <code>qualified</code> , then all attributes must be namespace qualified in the instance document. Note that if the <code>form</code> attribute is present on the <code>attribute</code> element, then it overrides <code>attributeFormDefault</code> on the schema element. All global attribute declarations must be qualified regardless of the value of the <code>form</code> attribute or <code>attributeFormDefault</code> attribute. For an attribute to be qualified in an instance document, it must have a prefix associated with the namespace.
blockDefault	#all   List of ( extension   restriction   substitution )	Enables you to block some or all of the derivations of data types from being used in substitution groups. The values <code>extension</code> and <code>restriction</code> block type substitutions, while the value <code>substitution</code> blocks element substitutions. This can be overridden by the <code>block</code> attribute of an <code>&lt;element&gt;</code> or <code>&lt;complexType&gt;</code> declaration in the schema.
elementFormDefault	qualified   unqualified	Enables you to specify a default value for element qualification in the instance document. If <code>qualified</code> , then all elements must be namespace qualified in the instance document. Note that if the <code>form</code> attribute is present on the <code>&lt;element&gt;</code> declaration, then it overrides <code>elementFormDefault</code> on the <code>&lt;schema&gt;</code> element. All global element declarations must be qualified regardless of the value of the <code>form</code> attribute or <code>elementFormDefault</code> attribute.
finalDefault	#all   List of ( extension   restriction   list   union )	Enables you to disallow some or all of the derivations of data types from being created in the XML Schema. This can be overridden by the <code>final</code> attribute of an <code>&lt;element&gt;</code> or <code>&lt;complexType&gt;</code> element in the schema.
id	ID	Gives a unique identifier to the element
targetNamespace	anyURI	This is used to specify the namespace that the schema is defining.
version	token	Used to specify the version of the XML Schema being defined. This can take a <code>token</code> data type, and is intended for use by XML Schema authors.
xml:lang	language	Specifies the language of the XML Schema being defined, using a code defined by RFC 3066. Most languages can be identified by a simple two-letter code.

## selector

The `<selector>` element is used when creating identity constraints, such as `<key>`, `<keyref>`, and `<unique>` declarations. When creating identity constraints, you must specify a context, or scope, for the constraint using a `<selector>` declaration, and the specific node that is constrained using a `<field>` declaration. It may contain an `<annotation>` element. For a complete example and further discussion on identity constraints, please see `key`. For more information, see §3.11.2 of the Recommendation.

### Example

```
<xs:key name="KeyDepartmentByID">
  <xs:selector xpath="example:Departments/example:Department" />
  <xs:field xpath="@departmentID" />
</xs:key>
```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
xpath (required)	XPath	A <i>relative</i> XPath expression (relative to the element on which the identity constraint is defined) that specifies to which elements the identity constraint applies

## sequence

The `<sequence>` element is used within content model declarations. It is used to declare a specific order of elements and content model declarations to be used within the content model in the instance document. It may contain `<annotation>` and `<element>` declarations. Because you can nest content models, it may contain `<choice>`, `<sequence>`, `<group>`, and `<any>` elements. Similarly, it can be contained by `<choice>`, `<group>`, `<sequence>`, or `<complexType>` elements. For more information, see §3.8.2 of the Recommendation.

### Example

```
<xs:sequence>
  <xs:element name="FirstName" type="xs:string" />
  <xs:element name="MiddleInitial" type="xs:string" />
  <xs:element name="LastName" type="xs:string" />
</xs:sequence>
```

## Appendix E: XML Schema Element and Attribute Reference

---

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
maxOccurs	nonNegative Integer or unbounded	The maximum number of times the model group can occur
minOccurs	nonNegative Integer	The minimum number of times the model group can occur

### simpleContent

The `<simpleContent>` element is used when extending or restricting complex types. It indicates that the resulting content model may contain attributes and text data, but cannot contain element content or mixed content. This element is used inside a `<complexType>` declaration and can contain an `<annotation>`, `<restriction>`, or `<extension>` element. For more information, see §3.4.2 of the Recommendation.

#### Example

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

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element

### simpleType

The `<simpleType>` element is used to specify the allowable type of content for attributes and text-only elements. Simple type definitions are the key to the validation of text content within XML Schemas. A `<simpleType>` declaration can be contained within an `<attribute>`, `<element>`, `<list>`, `<redefine>`, `<restriction>`, `<schema>`, or `<union>` declaration. It may contain `<annotation>`, `<list>`, `<restriction>`, or `<union>` declarations. For more information, see §3.14.2 of the Recommendation.

#### Example

```
<xs:simpleType name="FixedLengthString">
  <xs:restriction base="xs:string">
    <xs:length value="120" />
  </xs:restriction>
```

```

</xs:simpleType>

<xs:simpleType name="Size" >
  <xs:restriction base="xs:string" >
    <xs:enumeration value="S" />
    <xs:enumeration value="M" />
    <xs:enumeration value="L" />
    <xs:enumeration value="XL" />
  </xs:restriction>
</xs:simpleType>

```

### Attributes

Attribute	Value Space	Description
final	#all   List of (union   restriction)	Restricts how new data types may be derived from this simple type
id	ID	Gives a unique identifier to the element
name	NCName	The name of the data type that this element is defining. The name attribute is only used on global <simpleType> declarations.

### union

The <union> declaration enables you to join numerous simple data types together. You can include existing types in the union by referring to them within a whitespace-separated list in the `memberTypes` attribute. They are joined along with any contained <simpleType> declarations to form the new data type. The <union> declaration must be contained within a <simpleType> declaration and may contain <annotation>, or <simpleType> declarations. For more information, see §3.14.2 of the Recommendation.

### Example

```

<xs:simpleType name="CatsAndDogs">
  <xs:union memberTypes="CatBreeds DogBreeds" />
</xs:simpleType>

```

### Attributes

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
memberTypes	List of QName	A whitespace-separated list of simple data types that you want to join together to form a new <simpleType>.

### **unique**

The `<unique>` declaration enables you to specify that an element or attribute must have a unique value within a document or part of a document. The unique value might be the element content, a specific attribute's content, an ancestor's element or attribute content, or a combination of any of these options. You may specify the item that contains the unique value using the `<selector>` and `<field>` declarations. The `<unique>` element must be contained by an `<element>` declaration and may contain `<annotation>`, `<selector>`, or `<field>` declarations. For more information about identity constraints, please see `key`. For more information, see §3.11.2 of the Recommendation.

#### **Example**

```
<xs:unique name="employeeIdentificationNumber">
  <xs:selector xpath="example:Employees/example:Employee" />
  <xs:field xpath="@employeeID" />
</xs:unique>
```

#### **Attributes**

Attribute	Value Space	Description
id	ID	Gives a unique identifier to the element
name	NCName	The name of the identity constraint for the unique value being defined

## XML Schema Instance Attributes

The XML Schema Instance namespace is declared in an instance document to refer to instance-specific XML Schema attributes. (The namespace does not include any elements.) For example, the document can indicate to the parser the location of the schema to which it conforms using the `schemaLocation` attribute. The XML Schema instance namespace is: `http://www.w3.org/2001/XMLSchema-instance`, and is declared in the document element like this:

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

All the attributes detailed in the following table would be prefixed by `xsi:` as in the previous case.

Attribute	Value Space	Description
nil	boolean	Used to indicate that an element is valid despite having an empty value. Necessary for simple types, such as dates and numbers, for which empty values aren't valid:  <code>&lt;OrderDate xsi:nil="true"&gt;&lt;/OrderDate&gt;</code>

## Appendix E: XML Schema Element and Attribute Reference

Attribute	Value Space	Description
noNamespaceSchemaLocation	anyURI	Used to specify the location of a schema without a target namespace:  <code>xsi:noNamespaceSchemaLocation="name.xsd"</code>
schemaLocation	List of anyURI (in namespace / location pairs)	Used to specify the location of a schema with a target namespace. The namespace of the schema is specified first, followed by a space, followed by the location of the schema. Multiple namespace/location pairs can be provided as a whitespace-separated list:  <code>xsi:schemaLocation=" http://www.example.org/name example.xsd http://www.example.com/contacts contacts.xsd"</code>
type	QName	Enables you to override the current element type by specifying the qualified name of a type in an existing XML Schema. Note that the data type has to be derived from the one that the element is declared with. In addition, the substitution of the derived type cannot be blocked by the element or type declaration:  <code>&lt;returnAddress xsi:type="ipo:USAddress"&gt;</code>

