# LEARNING xsd

#xsd

# Table of Contents

About ... 1

Chapter 1: Getting started with xsd ... 2

Remarks ... 2

Versions ... 2

Examples ... 2

Installation or Setup ... 2

Chapter 2: xs:complexType ... 4

Introduction ... 4

Parameters ... 4

Remarks ... 5

Examples ... 5

Global ComplexType with Sequence and attributes ... 5

Creating a global xs:complexType by extending an existing xs:complexType ... 6

Creating a global xs:complexType by restricting an existing xs:complexType ... 7

Chapter 3: xs:schema ... 10

Introduction ... 10

Parameters ... 10

Examples ... 11

Basic xs:schema ... 11

xs:schema elementFormDefault attribute ... 12

Credits ... 15

# About

# Chapter 1: Getting started with xsd

## Remarks

XML Schema is a language and framework for validating XML documents.

An XML document that is **well-formed**, in the sense that it is syntactically conformant to the XML specification, can be tested for **validity** against a schema. The distinction between well-formedness, which is absolute, and validity, which is relative to a schema, is paramount.

Validation encompasses:

- Checking whether the XML document fulfils additional requirements such as the elements having certain names, restrictions on the content of elements, consistency constraints (primary keys, uniqueness, etc), attribute values or text matching certain types.
- Upon success, conversion of the input data model instance (called XML Infoset) to an output instance (PSVI: Post-Schema-Validation Infoset), where elements and attributes are annotated with type information, where default values have been populated, etc.

XML Schema was introduced to address requirements that DTD validation failed to address, among others a more complete type system including a rich set of builtin types, type restriction and extension capabilities, and more control on the restriction of element layout.

## Versions

| Version | Release Date |
|---|---|
| 1.0 | 2001-05-02 |
| 1.0, Second Edition | 2004-10-28 |
| 1.1 | 2012-04-05 |

XML Schema 1.0 was approved as a W3C Recommendation in May 2001, and the second edition incorporating errata was published as a W3C Recommendation several years later.

XML Schema 1.1 became a W3C Recommendation in 2012, which fixed more bugs and added other improvements, while being mostly compatible with earlier versions.

## Examples

### Installation or Setup

XSD, XML Schema Definition, is a language which describes the structure of XML documents. XSD files can be used to validate an XML file. The process of doing this will depend on what you choose to implement it with. Care should be taken to ensure the validation engine you use is compatible with the desired version of XSD.

Read Getting started with xsd online: https://riptutorial.com/xsd/topic/2907/getting-started-with-xsd

# Chapter 2: xs:complexType

## Introduction

A xs:complexType provides a description of an XML element's content in the instance document. The definition of the xs:complexType can be made globally in which case it has a name and can be re-used within the schema, or it can be inplace and only used within the context it is declared.

## Parameters

| Attributes | Description |
|---|---|
| abstract | When set to true the complex type can not be used directly in an instance XML document via xsi:type. It can however be used as the base type for an element definition. (default false) - Only valid for root level xs:complexType's |
| block | Limits the types that can be used in an XML instance document (defaults to the value of the xs:schemas blockDefault attribute if set, otherwise defaults to empty, values '#all' \| a list of ('extension', 'list', 'union') separated by whitespace). |
| final | Limits deriving types from using this type in certain ways within the schema (defaults to the value of the xs:schemas finalDefault attribute if set, otherwise defaults to empty, values '#all' \| or a list of ('extension', 'list', 'union') separated by whitespace) - Only valid for root level xs:complexType's |
| id | The id of the schema item (optional) |
| mixed | Indicates the instance XML element may contain mixed content (defaults to false) |
| name | The name of the xs:complexType - Only valid for root level xs:complexType's |
| any | Any other attributes not in the 'http://www.w3.org/2001/XMLSchema' namespace are allowed. |
| ----------------- | ------ |
| **Elements** | **Description** |
| ----------------- | ------ |
| xs:annotation | Provides the ability to add documentation and machine readable data. |

| Attributes | Description |
|---|---|
| xs:simpleContent | Used when the xs:complexType derives from a xs:simpleType. |
| xs:complexContent | Used when the xs:complexType derives from another xs:complexType. |
| xs:group | Adds the elements from an xs:group to the xs:complexType definition |
| xs:all | Adds the elements from an xs:all to the xs:complexType definition |
| xs:choice | Adds the elements from an xs:choice to the xs:complexType definition |
| xs:sequence | Adds the elements from an xs:sequence to the xs:complexType definition |
| xs:attribute | Adds the xs:attribute to the xs:complexType definition |
| xs:attributeGroup | Adds the xs:attributeGroup to the xs:complexType definition |
| xs:anyAttribute | Adds the xs:anyAttribute to the xs:complexType definition |

## Remarks

**Deriving from a xs:complexType**

When a xs:complexType derives from another xs:complexType is can do it via *extension* or *restriction*.

- extension - the deriving type takes everything defined in the base type and adds to it.
- restriction - the deriving type takes only selected parts from the base type, only allowing the parts it wants, no additional items can be added.

**Deriving from a xs:simpleType**

When a xs:complexType derives from a xs:simpleType is can do it via *extension*, in which case it can add attributes to the resulting type, but not elements.

**Content Type**

Conceptually a xs:complexType either contains *simple* or *complex* content. If the xs:complexType derives from a typed based on xs:anySimpleType (xs:int, xs:string etc) then it is *simple*. If it derives from a xs:complexType which contains *complex* content, then it itself is *complex* (if the xs:complexType does not derive from a type, then it is also complex).

## Examples

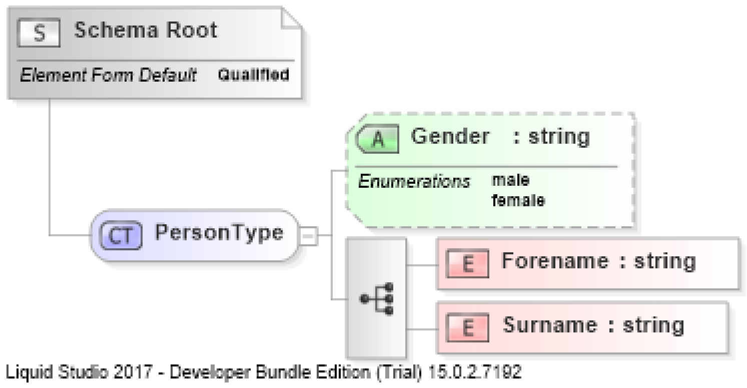
### Global ComplexType with Sequence and attributes

This example shows a simple global definition of a complexType. The definition is considered global as it is a child of the xs:schema. Globally defined types can be used elsewhere in the schema.

This is the most common form for declaring a global xs:complexType, it defines the child elements using a xs:sequence, xs:choice or xs:all, and optionally has attributes as well.

Note : because it is a globally defined it must have a unique name within the schema set.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="PersonType">
        <xs:sequence>
            <xs:element name="Forename" type="xs:string" />
            <xs:element name="Surname" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="Gender">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="male" />
                    <xs:enumeration value="female" />
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:schema>
```



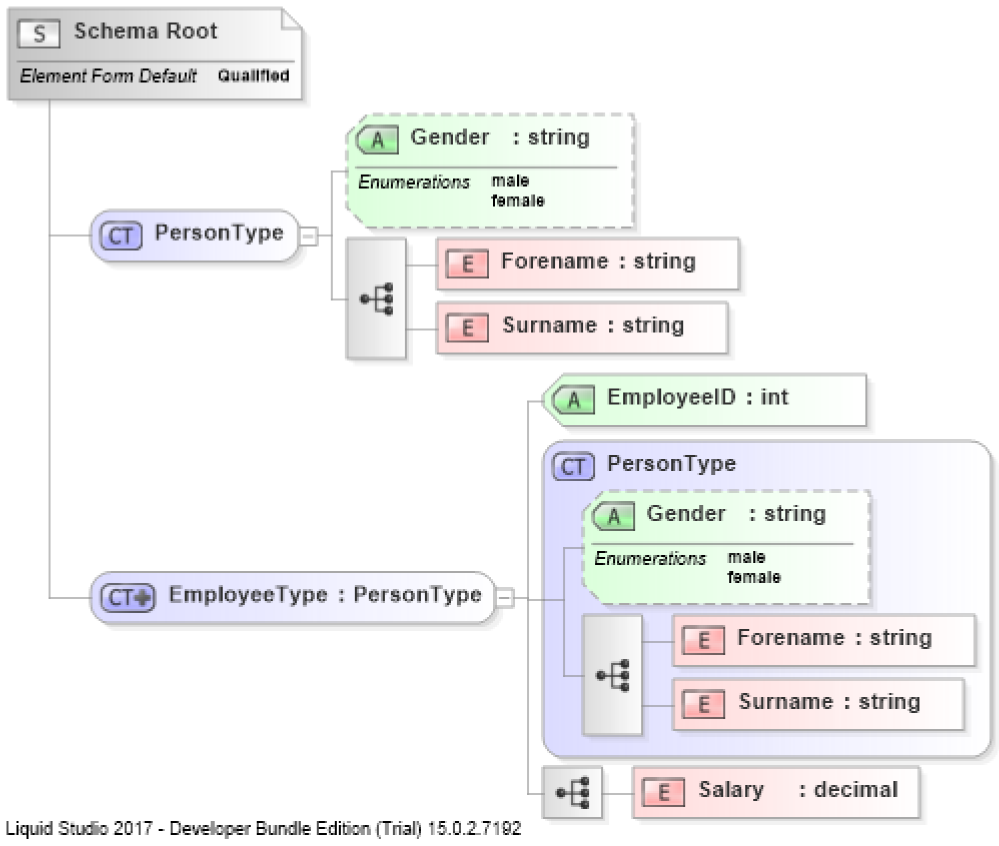
## Creating a global xs:complexType by extending an existing xs:complexType

In this example we are creating a new xs:complexType (EmployeeType) based on an existing xs:complexType (PersonType).

The construction of this is slightly more complicated. Because the base xs:complexType (PersonType) is considered to be *complex* (more about this below) we add the <xs:complexContent> element. Then because we are extending PersonType, we add the element <xs:extension base="PersonType">. Within the xs:extension tag we can add a compositor (xs:all/xs:choice/xs:sequence) and any additional attributes.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="qualified"
```

```
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="PersonType">
        <xs:sequence>
            <xs:element name="Forename" type="xs:string" />
            <xs:element name="Surname" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="Gender">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="male" />
                    <xs:enumeration value="female" />
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="EmployeeType">
        <xs:complexContent>
            <xs:extension base="PersonType">
                <xs:sequence>
                    <xs:element name="Salary" type="xs:decimal" />
                </xs:sequence>
                <xs:attribute name="EmployeeID" type="xs:int" use="required" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```



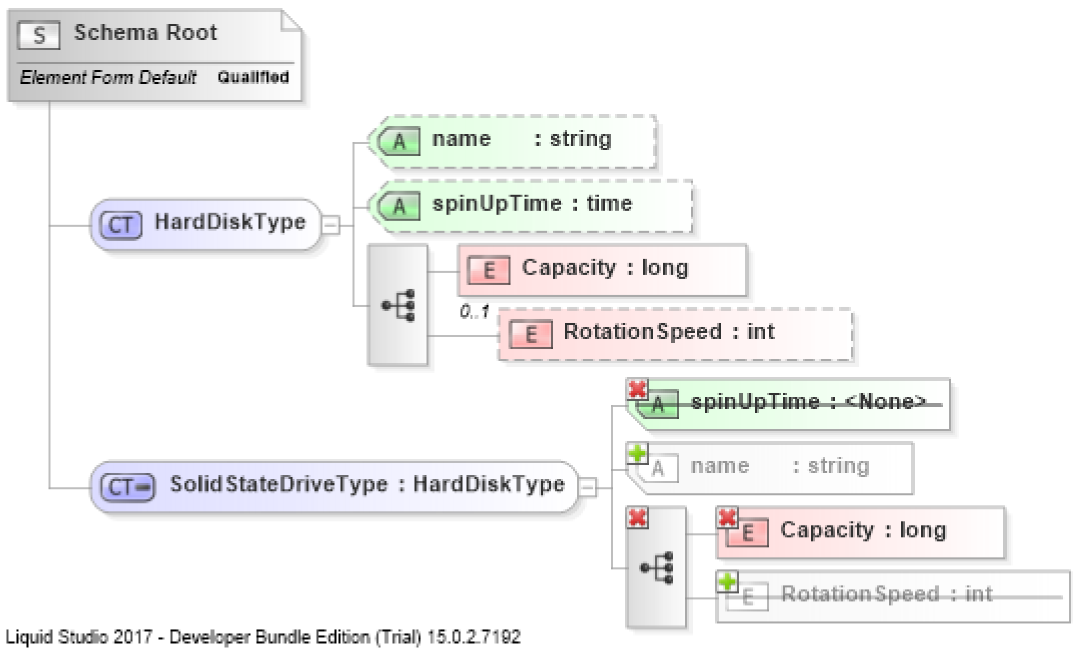
## Creating a global xs:complexType by restricting an existing xs:complexType

This is where things get a little tricky. We are now restricting an existing xs:complexType. Our SolidStateDriveType derives from HardDiskType but removes the spinUpTime attribute and the RotationSpeed element.

Notice the approach for dealing with attributes and elements is different. To remove an attribute you need to re-declare it and set its *use* to *prohibited*. For elements simply not re-declaring them will cause them to be excluded, in fact you need to re-declare any elements you want to keep in the new type.

***Key concept for restricted types** : It must be possible to load an XML instance element resulting from the restricted type into the base type, put another way the restricted type needs to be able to 'fit' into the base type. So you can not exclude a mandatory attribute or element, in order to exclude it in the restricted type it must be optional in the base type. If you change the type/facet rules of an element or attribute in the restricted type, the new type/facet rules must be compatible with the base type, so if the base type was a short, the restricted type could be a byte, but not a long.*

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 - Developer Bundle Edition (Trial) 15.0.2.7192
(https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="HardDiskType">
        <xs:sequence>
            <xs:element name="Capacity" type="xs:long" />
            <xs:element name="RotationSpeed" type="xs:int" minOccurs="0" />
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" />
        <xs:attribute name="spinUpTime" type="xs:time" />
    </xs:complexType>
    <xs:complexType name="SolidStateDrive">
        <xs:complexContent>
            <xs:restriction base="HardDiskType">
                <xs:sequence>
                    <xs:element name="Capacity" type="xs:long" />
                </xs:sequence>
                <xs:attribute name="spinUpTime" use="prohibited" />
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```


S Schema Root
Element Form Default Qualified
A name : string
A spinUpTime : time
CT HardDiskType
E Capacity : long
0..1
E RotationSpeed : int
A ~~spinUpTime : <None>~~
A name : string
CT SolidStateDriveType : HardDiskType
E Capacity : long
E ~~RotationSpeed : int~~
Liquid Studio 2017 - Developer Bundle Edition (Trial) 15.0.2.7192

Read xs:complexType online: https://riptutorial.com/xsd/topic/9047/xs-complextype

# Chapter 3: xs:schema

## Introduction

Describes for elements, attributes and types that are valid in an XML instance document. An XML Schema (XSD) must contain a single root level xs:schema element.

## Parameters

| Attributes | Description |
|---|---|
| attributeFormDefault | Indicates whether attributes in the XML instance document have to be qualified with a namespace (default unqualified) |
| blockDefault | The default value of the *block* attribute that is applied to xs:complexType and xs:element. Defines the rules for blocking derivation/substituion in the instance document (default empty, i.e. block nothing) |
| defaultAttributes | **(XSD 1.1)** Specifies an xs:attributeGroup that will be associated with all xs:complexType and xs:element within the schema (optional). |
| elementFormDefault | Indicates whether element in the XML instance document have to be qualified with a namespace (default unqualified). **Note** : almost without exception all schemas set this to '*qualified*'. |
| finalDefault | The default *final* attribute value used in xs:complexType and xs:element. Defines the rules for blocking derivation/substituion in the schema (default empty, i.e. block nothing) |
| id | The id of the schema item (optional) |
| targetNamespace | Qualifies all the elements and attributes (and globally defined components) defined within this schema. |
| version | The version of the schema, this is the version of the document, not this XSD version (i.e. Soap 1.2, FpML 4.2 etc) |
| xpathDefaultNamespace | **(XSD 1.1)** The default value for the the attribute *xpathDefaultNamespace* which is used in xs:selector, xs:field, xs:alternative & xs:assert (specifies the default namespace to be used in XPath expressions) |
| any | Any other attributes not in the ' http://www.w3.org/2001/XMLSchema' namespace are allowed. |

| Attributes | Description |
|---|---|
| **Elements** | **Description** |
| xs:annotation | Provides the ability to add documentation and machine readable data. |
| xs:include | Used to include a schema with the same targetNamespace, or no targetNamespace (see chameleon schemas). |
| xs:import | Used to include a schema with a targetNamespace different to the parent. |
| xs:redefine | Used to include a schema with the same targetNamespace (or no targetNamespace), and modify xs:simpleType, xs:complexType, xs:group or xs:attributeGroup definitions contained within it (here be dragons....) |
| xs:simpleType | Defines a global (named) simple type which can then be referenced and re-used. |
| xs:complexType | Defines a global (named) complex type which can then be referenced and re-used. |
| xs:group | Defines a global (named) group of elements which can then be referenced and re-used. |
| xs:attributeGroup | Defines a global (named) group of attributes which can then be referenced and re-used. |
| xs:attribute | Defines a global (named) attribute which can then be referenced and re-used. |
| xs:element | Defines a global (named) element which can then be referenced and re-used, or used as the basis of an XML instance document. |
| xs:notation | - |
| xs:defaultOpenContent | **(XSD 1.1)** Specifies rules for allowing additional elements to be permitted within every xs:complexType and xs:element within the schema. |

## Examples

### Basic xs:schema

Shows a very basic schema.

**Note:** by convention elementFormDefault is set to '*qualified*', in the really world you will be hard pressed to find a schema that does not set this (so just include it in your schemas!).

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 - (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Forename" type="xs:string" />
                <xs:element name="Surname" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

## xs:schema elementFormDefault attribute

By convention elementFormDefault is always set to *qualified*, but lets look at what it actually does.

---

First with elementFormDefault set to qualified.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="qualified"
           targetNamespace="http://base.com"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="MyBaseElement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ChildA" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Sample XML Document

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Studio 2017 (https://www.liquid-technologies.com) -->
<b:MyBaseElement xmlns:b="http://base.com"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://base.com ElementFormDefault_qualified.xsd">
    <b:ChildA>string</b:ChildA>
</b:MyBaseElement>
```

Notice the element ChildA must also be qualified with the namespace 'b'.

---

Now lets look at it with elementFormDefault set to unqualified.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="unqualified"
```

```
           targetNamespace="http://base.com"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="MyBaseElement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ChildA" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Sample XML Document

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Studio 2017 (https://www.liquid-technologies.com) -->
<b:MyBaseElement xmlns:b="http://base.com"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://base.com ElementFormDefault_unqualified.xsd">
    <ChildA>string</ChildA>
</b:MyBaseElement>
```

Notice this time that only the globally defined element MyBaseElement is qualified with the namespace 'b', the inner element ChildA (which is defined in place within the schema) is not qualified.

---

In the last example we saw that the globally defined elements must be qualified in the XML instance document, but elements defined inplace do not. But this does not just mean the root element, if you have globally defined elements that are referenced, then they need qualifying as well.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid Studio 2017 (https://www.liquid-technologies.com)-->
<xs:schema elementFormDefault="unqualified"
           targetNamespace="http://base.com"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="MyBaseElement">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ChildA" type="xs:string" />
                <xs:element xmlns:q1="http://base.com" ref="q1:MyElement" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="MyElement" type="xs:string" />
</xs:schema>
```

Sample XML Document

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Studio 2017 (https://www.liquid-technologies.com) -->
<b:MyBaseElement xmlns:b="http://base.com"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://base.com ElementFormDefault_unqualified.xsd">
    <ChildA>string</ChildA>
    <b:MyElement>string</b:MyElement>
</b:MyBaseElement>
```

Notice that MyElement also need qualifiying as it is globally defined.

---

In conclusion, if you have elementFormDefault set to qualified, then everything needs to be qualified with a namespace (either via a namespace alias or by setting the default namesapce xmlns="..."). However elementFormDefault is set to unqualified, things get complicated and you need to do some quite indepth examination of the schemas to work out if things should be qualified or not.

I assume that is why elementFormDefault is always set to qualified!

Read xs:schema online: https://riptutorial.com/xsd/topic/9052/xs-schema

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with xsd | Beth Whitezel, Community, Ghislain Fourny, whrrgarbl, Wolfgang Schindler |
| 2 | xs:complexType | Sprotty |
| 3 | xs:schema | Sprotty |