Enterprise Systems and Architecture

CMPU4025

XSD

XML Schemas

Schemas

- A **schema** is an XML document that defines the content and structure of one or more XML documents.
- Alternative to Document Type Definitions (DTDs)
- The XML document containing the content is called the **instance document.**

Schema Dialects

- There is no single schema form. Several schema “dialects” have been developed in the XML language.
- Support for a particular schema depends on the XML parser being used for validation.

Schema Dialects

This figure shows a few schema dialects

SCHEMA	DESCRIPTION	URL
DDML	DDML (Document Definition Markup Language) is a schema language for XML documents, encoding the logical (as opposed to physical) content of DTDs in an XML document. It is also known as XSchema.	*http://www.w3.org/TR/NOTE-ddml*
RELAX	The RELAX (Regular Language for XML) schema is based on a Japanese national schema standard.	*http://www.xml.gr.jp/relax*
RELAX NG	The RELAX NG schema combines the RELAX and NG schema specifications.	*http://www.oasis-open.org/committees/relax-ng/*
Schematron	The Schematron schema represents documents using a tree pattern, allowing support for those document structures that might be difficult to represent in more traditional schema languages.	*http://www.ascc.net/xml/resource/schematron/schematron.html*
TREX	The TREX (Tree Regular Expressions) schema specifies a pattern for the structure and content of an XML document, identifying a class of XML documents that match the pattern.	*http://www.thaiopensource.com/trex/*
XDR	The XDR (XML-Data Reduced) schema is developed and supported by Microsoft, in particular Microsoft's Internet Explorer browser. XDR is sometimes referred to as XML-Data.	*http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm*
XML-Schema	XML-Schema, created by the W3C Schema Working Group, is a large specification designed to handle a broad range of document structures. It is also referred to as XSD.	*http://www.w3.org/XML/Schema*

Starting a Schema File

- A schema is always placed in a separate XML document that is ***referenced*** by the ***instance document***.

Elements and Attributes of the Patient Document

This figure shows the elements and attributes of the Patient document

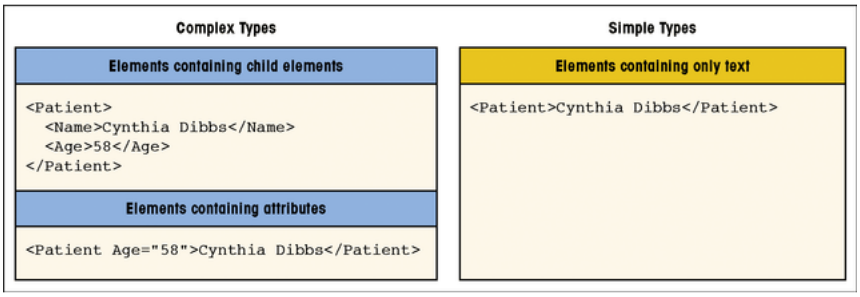
ELEMENTS	DESCRIPTION
Patients	The root element of Allison's document
Patient	The element that stores information about each individual patient (Name, ID, DOB, Stage, and Performance)
Name	The patient's name
ID	The patient's medical record number in the format MR###-###-##
DOB	The patient's date of birth in the format YYYY-MM-DD
Age	The patient's age (must be 21 or older)
Stage	The stage of the patient's breast cancer (must be either I or II)
Performance	A measure of the patient's health (must be a number between 0 and 1)
Comment	Optional comments providing additional information about the patient
ATTRIBUTES	
Scale	A required attribute of the Performance element indicating the type of performance measure; the attribute value must be equal to either "Karnofsky" or "Bell"

Schema (Element) Types

- XML Schema recognize two categories of element types: complex and simple.
- A **complex type** element has one or more attributes, **or** is the parent to one or more child elements.
- A **simple type** element contains only character data and **has no attributes**.

Schema Types

This figure shows types of elements



Simple Type Elements

- Use the following syntax to declare a simple type element in XML Schema:

  <element name="*name*" type ="*type*"/>

- Here, *name* is the name of the element in the instance document and *type* is the data type of the element.

- If a namespace prefix is used with the XML Schema namespace, any XML Schema tags must be qualified with the namespace prefix.

e.g.
<xs:element name="message" type="xs:string"/>

Understanding Data Types

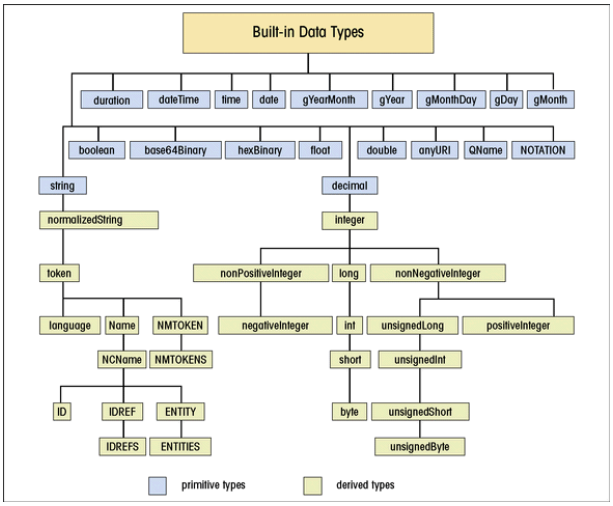
- XML Schema supports two data types: built-in and user-derived.
- A **built-in data type** is part of the XML Schema specifications and is available to all XML Schema authors.
- A **user-derived data type** is created by the XML Schema author for specific data values in the instance document.

Understanding Data Types

- A **primitive data type**, also called a base type, is one of 19 fundamental data types not defined in terms of other types.
- A **derived data type** is a collection of 25 data types that the XML Schema developers created based on the 19 primitive types.

Understanding Data Types

This figure shows the 44 built-in data types



Understanding Data Types

This figure shows a partial description of XML Schema data types

DATA TYPE	DESCRIPTION	EXAMPLES
string	Any legal XML text string	Cynthia Dibbs
decimal	A decimal number of arbitrary precision	3.14, 5.9E–10, 0, 7.0
integer	An arbitrarily large or small integer	0, 10, –10
positiveInteger	An integer strictly greater than zero	10
nonNegativeInteger	An integer greater than or equal to zero	0, 10
negativeInteger	An integer strictly less than zero	–10
nonPositiveInteger	An integer less than or equal to zero	0, –10
boolean	A value representing a binary outcome (0, 1, true, or false)	0, 1, true, false
date	A date in the format CCYY-MM-DD where CC represents the century, YY represents the year, MM represents the month, and DD represents the day	2003-04-01
ID, IDREF, ENTITY, ENTITIES, NMTOKEN, NMTOKENS	Derived data types based on the original DTD data types for attribute values	

Complex Type Elements

The syntax for complex type elements is:

<element name="*name*">
<complexType>
compositor
element declarations
compositor
attribute declarations
</complexType>
</element>

Complex Type Elements

- Here, *name* is the name of the element in the instance document
- *compositors* define how the list of elements is to be organized
- *element declarations* are simple type element declarations for each child element
- *attribute declarations* define any of the attributes of the elements.

Compositors

- A **compositor** is a schema tag that defines how the list will be treated. Three types of compositors are supported: sequence, choice, and all.
- The **sequence** **compositor** forces elements to be entered in the same order as indicated in the schema.
- The **choice** **compositor** allows any **one** of the items in the list to be used.
- The **all** **compositor** allows any of the items to appear in any order.
- Compositors may be nested inside of one another.

Declaring an Attribute

- Any element that contains an attribute is also a complex type. The syntax to declare an attribute is:

<attribute name="*name*" type="*type*" use="*use*"
default="*default*" fixed="*fixed*"/>

Attribute Examples

<xs:attribute name="lang" type="xs:string" **default="EN"**/>

If no value is specified in the XML instance the default as specified will be used

<xs:attribute name="lang" type="xs:string" **fixed="EN"**/>

The value "EN" will always appear

<xs:attribute name="lang" type="xs:string" **use="required"**/>

Use of the attribute is mandatory

Example – excerpt from note.xsd

```
<note id="12">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
     <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Attaching a Schema to a Namespace

- The syntax to associate the schema with a namespace is:

<*prefix*:schema xmlns:*prefix*=http://www.w3.org/2001/XMLSchema>

```
<?xml version="1.0"?>
<xs:schema   xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Structuring a Schema

- Schemas can be structured in a number of ways. One structure is called a **Russian Doll design.** This design involves sets of nested declarations.
- While this design makes it easy to associate the schema with the instance document, it can be confusing and difficult to maintain.

Russian Doll Design

This figure shows a Russian Doll design

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Schema for breast cancer patient data -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://uhosp/patients/ns"
            targetNamespace="http://uhosp/patients/ns">
  <xsd:element name="Patients">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Patient" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"/>
              <xsd:element name="ID" type="xsd:string"/>
              <xsd:element name="DOB" type="xsd:date"/>
              <xsd:element name="Age" type="xsd:positiveInteger"/>
              <xsd:element name="Stage" type="xsd:string"/>
              <xsd:element name="Comment" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="Performance">
                  <xsd:complexType>
                    <xsd:simpleContent>
                      <xsd:extension base="xsd:decimal">
                        <xsd:attribute name="Scale" type="xsd:string" use="required"/>
                      </xsd:extension>
                    </xsd:simpleContent>
                  </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Structuring a Schema

- Another schema design is a **Flat Catalog Design**.
- In this design, all element declarations are made globally.
- The structure of the instance document is created by referencing the global element declarations.
- The syntax is:

<element ref="*name*">

Flat Catalog Design

This figure shows a Flat Catalog design

Named Schema Elements

- A structure can be named creating a **named complex type** that can be used elsewhere in the schema.
- This has the advantage of "storing" the structure so it can be applied to the ref attribute.

Named Model Groups

- A **named model group** is a collection of elements. The syntax is:

  <group name="*name*">
  element declarations
  </group>

- The element declarations must be **enclosed within a sequence, all or choice compositor**.

Named Attribute Groups

- Attributes can be placed within **named attribute groups**. The syntax is:

```
<attributeGroup name="name">
  attribute declarations
</attributeGroup>
```

- This can be useful if you want to use attributes with several different elements in the schema.

Deriving New Data Types

- Three components are involved in deriving new data types:
  - **Value space**: the set of values that correspond to the data type.
  - **Lexical space**: the set of textual representations of the value space.
  - **Facets**: the properties of the data type that distinguish one data type from another.

User Derived Data

- New data types fall into three categories:
  - **List**: a list of values where each list is derived from a base type.
  - **Union**: the combination of two or more data types.
  - **Restriction**: a limit placed on the facet of a base type.

Deriving a Restricted Data Type

- The most common way to derive a new data type is to restrict the properties of a base type. XML Schema provides twelve **constraining facets** for this purpose.

Restriction Example

```
<xs:element name="age">
    <xs:simpleType>
     <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
     </xs:restriction>
    </xs:simpleType>
</xs:element>
```

This element is defined to have an integer value between 0 and 120 inclusive

Constraining Facets

This figure shows the 12 constraining facets

FACET	DESCRIPTION
length	Specifies the length of the datatype; for text strings, length measures the number of characters; for lists, length measures the number of items in the list
minLength	Specifies the minimum length of the datatype
maxLength	Specifies the maximum length of the datatype
pattern	Constrains the lexical space of the datatype to follow a specific pattern of characters
enumeration	Constrains the datatype to a specific set of values
maxInclusive	Specifies an upper bound for the datatype (can be used with datatypes that can be ordered, such as numbers); the upper boundary is included as a legitimate value
minInclusive	Specifies a lower bound for the datatype; the lower boundary is included
maxExclusive	Specifies an upper bound for the datatype, but the upper boundary is not included
minExclusive	Specifies a lower bound for the datatype, but the lower boundary is not included
whiteSpace	Controls the use of blanks in the lexical space; the whiteSpace facet has three values: preserve (no changes made to the content), replace (replace all tabs, carriage returns, and line feed characters with spaces), and collapse (replace all tabs, carriage returns, and line feeds, remove any opening or closing blanks, and collapse multiple blank spaces to a single blank space)
totalDigits	Constrains the value space to a maximum number of decimal places
fractionDigits	Constrains the value space to a maximum number of decimal places in the fractional part of the value

The Patterns Facet

- A pattern can be created with a formatted text string called a **regular expression** or **regex.**
- The basic unit of a regex is called an **atom**. It can be a single character, a group of characters, or another regex enclosed in parenthesis.
- A **quantifier** can be added to the atom to specify the number of occurrences for a particular character.

Pattern Quantifiers

This figure shows pattern quantifiers

QUANTIFIER	DESCRIPTION
?	Zero or one occurrence
*	Zero or more occurrences
+	One or more occurrences
{*min*, *max*}	Between *min* and *max* occurrences
{*n*}	Exactly *n* occurrences
{*min*, }	At least *min* occurrences

Annotating a Schema

It is helpful to include comments about a created schema for other XML developers. An annotation element stores information about the schema. The syntax is:

```
<annotation>
  <documentation>
    documentation comments
  </documentation>
  <appinfo>
    application information
  </appinfo>
</annotation>
```

Examples (ref: w3schools.com)

Example – note.xml

Here we specify a default namespace – all child elements will also belong to this namespace

```
<?xml version="1.0"?>

<note xmlns="http://www.w3schools.com">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
<note>
```

Example – note.xml

```
<?xml version="1.0"?>

<note xmlns="http://www.w3schools.com"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="note.xsd">

  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
<note>
```

Here, we also specify the xsi namespace prefix and use it to give the XML schema location\name

Example – note.xsd

```
<?xml version="1.0"?>
<xs:schema      xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.w3schools.com"
                xmlns="http://www.w3schools.com"
                elementFormDefault="qualified">

<xs:element name="note">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="to" type="xs:string"/>
   <xs:element name="from" type="xs:string"/>
   <xs:element name="heading" type="xs:string"/>
   <xs:element name="body" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

xmlns:xs specifies the namespace for the schema element names we use in the XSD file

The targetNamespace specifies the namespace that the elements we're designing will belong to

The xmlns specifies that the target namespace is the default namespace for this XSD file (which itself is XML).

```
//// Load XML file...
var doc = new ActiveXObject("MSXML2.DOMDocument.6.0");
doc.async = false;
doc.validateOnParse=true;
doc.load("notes_with_reference.xml");

/// Load Schema file...
var nameSpace = doc.documentElement.namespaceURI;
var xmlSchema = new ActiveXObject("MSXML2.XMLSchemaCache.6.0");
xmlSchema.add(nameSpace,"note.xsd");
doc.schemas = xmlSchema;

/// Parse/Validate XML file...
doc.load("notes_with_reference.xml");
var error = doc.parseError;
if (error != "")
{
alert(error.reason);
}

/// Write HTML output...
document.write("XML Root Tag Name: " + doc.documentElement.tagName + "<br/><br/>");

//Use the following…
//rootChildren = doc.documentElement.childNodes
//rootChildren[i].firstChild.nodeValue
```