Exercise Sheet 1 (SS 2023)

3.0 VU Semistrukturierte Daten

Information on the Exercise Sheet

General

In the first exercise sheet you will develop a schema for a file format. First, you will create an XML-Schema (XSD) and write a matching XML document. In the second part, you will convert the XML-Schema into a Document Type Definition (DTD).

Submit your solution as a single ZIP file (max. 5MB). The ZIP file should contain an XML-Schema, a DTD as well as fitting XML documents for the XML-Schema and the DTD. Thus, you should submit the following files:

- `shipment.xsd`
- `shipment-xsd.xml`
- `shipment.dtd`
- `shipment-dtd.xml`

The exercise sheet is made up of 5 tasks (described below). You can earn a total of 10 points.

Deadlines

at the latest April 18th	**12:00 Uhr**	Upload your solutions in TUWEL
Please do not forget	$\Longrightarrow$	Register for an exercise interview in TUWEL

Exercise Interviews

During the solution discussion, the correctness of your solution as well as your understanding of the underlying concepts will be assessed.

The scoring of your submission is primarily based on your performance at the solution discussion. Therefore, it is possible (in extreme cases) to receive 0 points although the submitted solution was technically correct.

Please, be punctual to your solution discussion. Otherwise, we cannot guarantee that your full solution can be graded in your assigned time slot. Remember to bring your student id to the solution discussion. It is not possible to assess your solution without an id.

Office Hours (optional)

Before the submission deadline we offer office hours with our tutors. If you have questions or problems with the material of the exercise sheet, you can get personal support during these office hours.

The aim of the office hours is to help your **understanding of the material** and not to check your solutions or to solve the exercise fo you.

Attendance is completely optional – times and locations for the office hours will be published on TUWEL.

Further Questions – TUWEL Forum

If you have any further questions, regarding organization or the material, you can use the TUWEL forum.

Exercises: XML Schema

First, you will write an XML schema for managing shipments in a supply chain. Save the created XML-Schema in the `shipment.xsd` file. In task 3 you will then be asked to create an XML document `shipment-xsd.xml` that matches the schema.

Important: Make sure that your schema file is well-formed and that your XML document is valid! If this is not the case you will receive 0 points for the associated tasks! If you have trouble implementing all aspects of the schema you still have to make sure that your schema is well-formed and the document is valid.

Aufgabe 1 (Defining the Elements of `shipment.xsd`) *[4 Punkte]*

The XML-Schema shall validate XML documents with the following structure:

Element `shipment`. The root element `shipment` stores all the shipment information of the supply chain and contains the following elements in any order *exactly once*:

- A `ships` element;
- a `products` element;
- a `tags` element.

Element `ships`. The `ships` element has no attributes. This subtree stores all the ship information of our supply chain. To do so, the `ships` element contains an arbitrary number of `ship` elements.

Element `ship` (Child of `ships`). A `ship` element has the following required elements in the following order: a `name` stored as a string as well as elements `info` and `tags` as described below. Furthermore, a `ship` element has a required attribute `sid` using an appropriate type for an identifier.

Element `info` (Child of `ship`). An `info` element is empty and has three attributes:

- A *required* date attribute `firstTour` (the date of the ship's first tour);
- an *optional* date attribute `lastTour` (the date of the ship's last tour, if there is one);
- and an *optional* string attribute `placeOfConstruction`.

Element `tags` (Child of `ship`). The `tags` element contains a list of tags associated with the ship. In particular, it contains between 0 and 4 (inclusive) `tag` elements. A `tag` element solely contains a string.

Element `products`. The `products` element has no attributes. This subtree stores all products known to our supply chain. To do so, the `products` element contains *at least 3* `product` elements.

Element `product` (child of `products`). A `product` element contains the following elements in any order:

- A *necessary* `name` element stored as a string;
- a *necessary* `type` element;
- a *necessary* `label` elements;
- a *necessary* `catalog` element;

- and an *optional* `tags` element which has the same type as the `tags` child of a `ship` element.

Element `type`. The type element describes the kind of the product. It contains exactly one `food` or one `clothing` element.

Element `food` A `food` element is empty and contains two *necessary* attributes:

- The attribute `foodType` can be one of the strings "Fruits", "Wine", or "Meat".
- The `storageInfo` attribute is a *non-empty* string.

Element `clothing` A `clothing` element is empty and contains one *optional* string attribute `material`.

Element `label`. The `label` element contains a textual description of the product. In the textual description there has to exist a `producer` element that contains a string marking the product's producer. Furthermore, there can be arbitrarily many `destination` elements that contain strings and arbitrarily `ref` elements as specified below.

Element `ref`. A `ref` element contains a string and has two *optional* string attributes `sid` and `t`.

Element `catalog`. A `catalog` element contains a string that adheres to the following format: First there is one upper case letter, followed by a hashtag ("#"), followed by four digits (0 to 9), followed by another hashtag, and finally three lower case letters in the range of "a-g". For example: "P#1234#aeg".

Element `tags` (child of `shipment`). The `tags` element directly below the root has no attributes. It contains an arbitrary number of `t` elements.

Element `t`. A `t` element contains a string describing the tag as well as a string attribute `tagname`.

Aufgabe 2 (Define keys and references `shipment.xsd`) *[2 Punkte]*

Add the following keys to your schema:

- A global key `shipKey` for the attribute `sid` of `ship` elements.
- A global key `tagKey` for the attribute `tagname` of `t` elements.
- A global key `prodKey` over `name` *and* `catalog` elements of `product` elements.

Now add the following key references to your schema:

- The `sid` attribute of `ref` elements references the `shipKeys`.
- The content of a `tag` element references the `tagKey`.

Finally, add the following uniqueness constraint to your schema:

- Every `ship` and `product` element contain an element `tags` which contains up to 4 `tag` elements. Make sure that *locally* in every such `tags` node, there are no duplicate entries in its `tag` child elements. For example, the following example would violate the constraint:

```
<tags>
 <tag>Eduard Green</t>
 <tag>Eduard Green</t>
 <tag>Thomas White</t>
</tags>
```

  But it is allowed that "Eduard Green" occurs as the content of a child of different `tags` elements.

Aufgabe 3 (Creation of the XML document `shipment-xsd.xml`) *[1 Punkte]*

Create the XML document `shipment-xsd.xml` for the schema `shipment.xsd`. The XML document shall satisfy the following conditions:

- Create at least 4 `ship` elements.
- Additionally, create at least 4 `product` elements, such that each child element of the `type` element (i.e., `food` and `clothing`) and each value of the `foodType` attribute (i.e., "Fruits", "Wine", and "Meat") occur at least once.
- Create at least 10 `tag` elements overall.
- Create at least 4 `t` elements.
- Create a `label` element that contains in its textual content at least 3 `destination` and at least 2 `ref` elements.

Make sure that your XML-Schema `shipment.xsd` validates your `shipment-xsd.xml` document. You can check this via the following command (after installing `xmllint`):

```
xmllint --schema shipment.xsd shipment-xsd.xml
```

Instructions for downloading and installing `xmllint` can be found in TUWEL.

Important: If your XML document is not well formed and valid with regards to the schema, you will receive 0 points for this task!

Document Type Definition

Now create a DTD for the above specification.

Aufgabe 4 (Creating a DTD `shipment.dtd`) *[2 Punkte]*

Create a Document Type Definition (DTD) `shipment.dtd`, which realizes the specification form Exercise 1 and 2 above. It is possible that parts of the specification are very complicated or impossible to implement in DTD. In that case make reasonable assumptions and adaptions to implement them as close as reasonable in the DTD.

In your exercise interview you have to be able to explain which parts are not fully realizable in a DTD (and why). **Important:** It is particularly important that you try to implement the key and key references. On the other hand it is not necessary to explicitly implement large

number ranges through explicit enumeration of all numbers (e.g., you do not have to create an enumeration of the numbers 1 through 72 to implement a "number between 1 and 72"). Enumerations that have a small range of up to 6 values should however be implemented fully in the DTD.

Important: If you submit a DTD with syntax errors, meaning it cannot be used for validation, you will receive 0 points for the task.

Aufgabe 5 (Creating the XML document `shipment-dtd.xml`) *[1 Punkte]*

If your previous XML document `shipment-xsd.xml` does not validate with your DTD, you will have to create an additional XML document `shipment-dtd.xml`, which contains the same data but validates for your DTD.

Make sure that your DTD `shipment.dtd` validates your XML document `shipment-dtd.xml`. You can check this using the following command (after installing `xmllint`):

```
xmllint --dtdvalid shipment.dtd shipment-dtd.xml
```

Important: If your XML document is not well-formed or valid with regards to your DTD, you will receive 0 points for this task.