XSD

Overview

XSDL (XML Schema Definition Language) allows to formally describe the elements which will compose a class of XML documents.
An XSD document is nothing more than another XML document[1], composed by multiple tags that defines how other XML document should be composed.
The produced XSD files can be then used to verify the validity of an XML instance with respect to a defined XML-based language.

Syntax

Elements: the building blocks of the XML-language we're defining, they contain the data and determine the structure of the document.
In an XSD document they can be defined as follows:
`<xsd:element name="species" type="xsd:string"/>`.
In this particular case we've defined that the XML document can contain an element called "species" of type "string".

Simple types: XSD offers the following simple types as a base to build more complex one.

- string
- decimal
- integer
- boolean
- date
- time

There are two ways to define a simple type element:

- Simple, in-line, notation:
  ```
  <xsd:element name="experience" type="xsd:integer"/>
  ```
- Nested notation that allows for particular kind of restrictions (e.g. just integers between [1-100]):
  ```
  <xsd:element name="experience">
        <xsd:simpleType>
              <xsd:restriction base="xsd:integer">
                    <xsd:minInclusive value="1"/>
                     xsd:maxInclusive value="100"/>
              </xsd:restriction>
        </xsd:simpleType>
  </xsd:element>
  ```

[1] In all these examples we're going to assume that under the namespace "xsd" is defined the reference to the XSDL schema definition `http://www.w3.org/2001/XMLSchema`

Complex types: a complex type contains other elements and/or attributes.
A Pokèmon's effect, as defined in pokemon.xsd, is an example of a complex type, since it's composed as it follows:

```
<xsd:complexType name="effect">
      <xsd:sequence>
            <!--Name of the effect→
            <xsd:element name="name" type="xsd:string"/>
           <!--Type of the effect (e.g. WATER, ELECTRIC,...)-->
           <xsd:element name="type" type="xsd:string"/>
        </xsd:sequence>
</xsd:complexType>
```

The example above defines how an effect should be presented in the XML file. It is possible to see that an "`effect`" contains a name and a type, where both are strings.

But I can also create some complex types built upon other complex types, such as a `Team` of Pokémon which is a collection of max 6 Pokèmon.

```
<xsd:complexType name="TeamType">
      <xsd:sequence>
         <!--A team can have at most 6 pokémon-->
         <xsd:element name="Pokemon" maxOccurs="6" type="pkm:PokemonType" />
      </xsd:sequence>
</xsd:complexType>
```

During the definition of complex types, some indicators are used to put constraints defining how a complex type is composed. These indicators are:

- **all** - specifies that the child elements can appear in any order, but has to appear once
- **sequence** - specifies that the child elements must appear in the specific order they're defined
- **choice** - specifies that either one child element or another can occur

Important attributes to keep in mind while using these iterators, are **minOccurs/maxOccurs** which specifies the minimum/maximum number of times an element can occur (note that all, sequence and choice already present some default value for them and if they're not overwritten you have to stick to them).

Important header attributes:
When using `<schema>` in order to define a new XSD schema, a number of different header attributes could be of vital importance, as can be seen trying to delete/modify them in pokemon.xsd and then performing validation again:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="pokemonNamespace" xmlns:pkm="pokemonNamespace"
      elementFormDefault="qualified">
```

Basically `xmlns:xsd="http://www.w3.org/2001/XMLSchema"` and `xmlns:pkm="pokemonNamespace"` are the common namespace we're already familiar with when we talk about XML documents (at the end XSDL is still an XML-based language itself): they're used to explicitly attach an element to its definition, a mean of disambiguation.
The `targetNamespace` will give a name to the scope we're creating within the content of `<schema>`, so that each of its children element we're going to define could be attached to it (when we're referring to them).
The `elementFormDefault` attribute set to `"qualified"` means that all the childrens element of the complex type element we're defining has still to be attached to a namespace.

References

If you're interested in more examples take a look at the XSDL documentation here https://www.w3.org/TR/xmlschema11-1/