Object Oriented Design: XML

Table of Contents

1 Lab 1

During this class you have to:

- perform analysis of data we are going to store and design XML-based file format that is capable of storing such data,
- write XSD (XML Schema Definition) defining the file format in a formal way,
- use Visual Studio to edit XML and configure is to use XSD to provide validation and suggestions,
- generate C# code containing classes representing XML structure,
- configure C# project so classes from XML are generated as an automatic compilation step,
- deserialize XML in C# program, check its validity and process results of deserialization.

The files crated during this class will be needed next week.

1.1 The data we are going to store

Assume that you own a library (in a sense of place where books are) and in the library you have books. Each book has:

- title,
- year,
- a language it is written in,
- one or more authors, each author has:
  - one or more names,
  - exactly one surname.

We want to store those information in some file.

/Note: During today's class we are not references between books and authors. (This will be done next week). Let every book contain all (possibly redundant) information about its authors./

1.2 Designing XML Structure

When designing XML-based format you should consider the following:

- XML can contain only single root. Is there any candidate for root of our structure or do we have to crate an artificial one?
- Which entities will be elements or sub-elements and which will be attributes?
  - Attributes must be of simple type, elements can be complex
  - Elements are easier to extend (useful if we expect changes to initial requirements).
  - There can be only one attribute of given name in an element.
  - Attribute syntax is more compact.
  - Is the structure coherent? (for example: if author's surname is an attribute, the author's name shouldn't be an element).

In design phase it would be sufficient to perform this process as a pure thought, but usually it is better to take some notes: we can create some XML storing exemplary data and make changes to it as we design it.

Think of some small library consisting of 2–3 books, one with one author, one with two authors, each in different language

1.2.1 Writing an XML for example data

To edit XML file use Visual Studio 2017.

1. Instead of creating a new project (we will do it later), create just a new file, choose XML as file type. A new XML file containint an XML header is created.
    1. Write contents of your example library in this file as XML.
    2. Verify if the proposed structure is sufficient for all cases. (e.g. can we store an author who has three names)?

1.2.2 Writing XSD (XML Schema definition)

Once you designed XML file structure, you have to write XSD file formally describing it.

Create new file of XML Schema (XSD) type in Visual Studio. Choose to edit source code with XML editor.

The file contents initially should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
</xs:schema>
```

Starting from this point you need to carefully examine namespaces. Reminder:

- `xmlns:prefix=name` means that in XML document all elements with in a form `prefix:element` are from namespace `name`
- `xmlns=name` defines namespace for elements without a prefix.

Mapping of the namespace to a prefix is local for document subtree. *It is possible to map namespace X to prefix a and in one document and namespace Y to prefix a in another one. You should always check namespace mappings in a document.*

XML Schema is also an XML-based language. Its elements are defined in a namespace. The namespace for XML Schema is `http://www.w3.org/2001/XMLSchema`. Visual Studio created a document that already contains mapping of that namespace to xs prefix. We can, for instance, create perfectly valid schema by mapping the namespace without a prefix, like in this document: http://mini.pw.edu.pl/~karwowskij/projob/xml/task1/bookstore.xsd

Besides XML Schema we have another namespace referenced three times:

- targetNamespace attribute – here we define the name of namespace that will be defined by this schema.
- mapping of the namespace we create to `mstns` and no prefix. This mapping is required to use any internal references between types or element within the schema.

Modify provided template:

- remove mapping to namespace without a prefix to avoid any name confusions,
- change target namespace to something reasonable like `http://example.org/jk/library`. Update the mapping of the namespace and change `mstns` prefix to something reasonable.

Make sure you have not removed `elementFormDefault="qualified` attribute. It states that all elements in this schema will be members of target namespace. In all typical cases this is what you want.

The resulting document should look similar to this one:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://example.org/jk/library"
    elementFormDefault="qualified"
    xmlns:lib="http://example.org/jk/library"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
</xs:schema>
```

Within `schema` element define the structure of your XML. Initially define the root element, like this:

```
<xs:element name="XXX">
 ....
</xs:element>
```

Now define a type of the root element. It can be done in two ways:

- use type attribute to refer to a type defined somewhere else or
- define an anonymous type within element.

```
<!-- element of type defined somewhere else -->
<xs:element name="XXX" type="xs:string"/>
```

Note: `<!--` and `-->` are XML comment delimiters. Text within those delimiters will be ignored.

```
<!-- element with anonymous type -->
<xs:element name="XXX">
 <xs:complexType>
   ....
```

```
 </xs:complexType>
</xs:element>
```

There are two kinds of typed in XML Schema:

complexType
: type that consists of attributes and sub-elements. May not be used as a type for attributes.

simpleType
: simple type (string, integer, …) or a restriction or union of such.

1. complexType definition

   complexType, in principle, defines two aspects:

   - child elements allowed in the element,
   - allowed and required attributes of the element.

   Example type:

   ```
   <xs:element name="XXX">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="A" type="xs:string" minOccurs="0" maxOccurs="3"/>
       <xs:element name="B" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
       <xs:element name="C" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="X" type="xs:integer" use="required"/>
    </xs:complexType>
   </xs:element>
   ```

   Inside the `complexType` there must be exactly one of the following:

   sequence
   : elements defined within mus appear in exactly that order. Each element must appear between minOccurs i maxOccurs (inclusive) times.

   all
   : defines that elements within can appear in any order, no more than once. We can make element mandatory by setting minOccurs to 1.

   choice
   : Exactly one of elements must appear. `xs:element` s an example of such as it can contain either `xs:complexType` or `xs:simpleType`, but not both at once.

   group
   : is used to named group of elements defined somewhere else. It will be used next week.

   After defining child elements in complexType arbitrary number of `attribute` elements can appear. Attribute element, besides name and type has use attribute. use="required" means that attribute is mandatory.

   We can nest arbitrary number of complexType elements:

   ```
   <xs:element name="XXX">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="A" minOccurs="0" maxOccurs="3">
        <xs:complexType>
          <xs:sequence>
           <xs:element name="B">
            <xs:complexType>
             ...
            </xs:complexType>
           </xs:element>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
    </xs:complexType>
   </xs:element>
   ```

   By nesting we can express any tree structure of finite depth.

1.2.3 Connecting the namespace with the document

Write XML Schema describing the format designed earlier.

When XSD is ready modify example XML document in a way that all elements without a prefix are from the defined namespace by adding `xmlns` attribute to document root.

The result should look similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<library xmlns="http://example.org/jk/library">
 ...
```

1.2.4 Visual Studio and XSD

Every XML processing tool has a base (usually called a catalog) of all namespaces known to it. The tool will use this definitions to perform various tasks. For instance XML editor in visual studio can:

- underline syntax errors,
- provide code completion,

based on document schemas (you have already seen it during XSD edition).

To add custom xsd in Visual Studio choose XML menu (available when editing XML) and choose `Schemas...` here you can configure list of namespaces known to XML editor. In recent Visual Studio versions if you edit XSD it is automatically added to this list. If it was not added you have add it manually here.

Once visual studio is configured check if you have:

- code completion and
- validation error reporting

when editing your xml format.

1.2.5 Named types: avoiding deeply nested schemas

We can define named types in XML Schema to avoid deep levels of nested element and to reuse types in several places:

```
<xs:schema targetNamespace="A" xmlns:a="A" ...>
 <xs:element name="XXX" type="a:typeX">
 </xs:element>
 <xs:complexType name="typeX">
  <xs:sequence>
   <xs:element name="A" minOccurs="0" maxOccurs="3" type="a:typeY">
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="typeY">
  ...
 </xs:complexType>
</xs:schema>
```

Named types appear after element definition in root schema element. The can be both `complexType` and `simpleType`. Any number of named typ definitions can appear. Each type must have `name` attribute set. Then the name can be referred by `type` attribute of an element.

Namespaces:

- all type definitions appear in `targetNamespace`,
- when defining type name we use a name without a prefix,
- all references to the type must contain namespace prefix.

1.2.6 Generating XSD file from XML instance

There are tools that are able to capture the structure of existing XML file and generate XSD based on some assumptions about types. One of such tools is `xsd.exe` which is one of the tools provided with Visual Studio. The program has several features regarding xsd files.

Interface of `xsd` is significantly better tans Visual Studio's (it's the text interface). To run xsd you have to start `Developer Command Prompt for VS 2017` and then run xsd, providing input xml and output directory as an arguments:

```
xsd.exe file.xml /outputdir:directory
```

Once file is generated you can examine it in Visual Studio. There are several issues:

- the structure is nested deeply,
- it contains additional useless namespace `xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"`,
- it probably haven't guessed some types right as well as number of element occurrences.

We can use such file as starting point instead of starting from empty file, but remember to:

- remove msdata namespace references?
- fix all improperly guessed types,
- check if all needed namespaces are properly mapped,
- check if `elementFormDefault="qualified` is present, add it if needed.
- check if `attributeFormDefault="qualified` is not present. If it is change value to unqualified.

You are allowed to use `xsd.exe` during third lab.

xsd manual: https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-schema-definition-tool-xsd-exe

1.2.7 Reference exact xsd file from xml

XML allows to refer to particular namespace definition directly from xml document, without specific configuration changes to xml catalog in xml tools. It is mainly useful during development when you frequently change xsd file or have several modified versions.

`XML Schema Instance` (XSI) allows us to provide path to xsd file next to xmlns mapping:

```
<?xml version="1.0" encoding="utf-8"?>
<library xmlns="http://example.org/jk/library"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://example.org/jk/library library.xsd"
>
```

```
...
</library>
```

Used syntax:

- inclusion of XSI namespace `http://www.w3.org/2001/XMLSchema-instance`, usually with xsi prefix
- attribute `schemaLocation="namepspace path"` tells XML processor that definition of `namespace` is available in file at given path.

Write two files describing your namespace, one under which your example document is valid and one under wthich the document is invalid. Use XSI do refer to particular XSD file and compare errors reported by Visual Studio.

1.3 C# classes from XSD

We are going to write a program that deserializes XML into objects and then processes it. We are going to create a solution consisting of two projects:

- a library, that contains classes used to deserialize xml and helper functions,
- a console application that uses classes from the library to read XML and then processes it.

First create a C# Class Library project (.NET Framework). Once project is created in `Solution Explorer` window you will see a project tree. First element root is called a solution – an aggregate of projects, the first and currently the only child is a class library project. We want to add another project to our solution. Press third (usually right) mouse button on the solution icon and choose to add new project. Choose C# Console Application.

After this operation you have a solution consisting of two independent projects. We are going to use classes defined in class library in the console application. To do so we must add new reference to Console Application project: click *add reference*. In add reference window choose solution in the left pane and choose your class library project. After this step you will be able to use classes from Class Library project in Console Application.

Press third mouse button on Console Application and choose Set as Startup project.

1.3.1 xsd.exe – code generation

We can use `xsd.exe` to generate classes that can represent our XML structure.

```
xsd.exe /c /n:Library /out:directory file.xsd
```

This command will generate C# classes from provided xsd file. Option `/n` defines namespace in which generated classes will be put. Option `/out` specifies output directory fo generated files.

You can now examine generated C# files.

1.3.2 Integrating class generation with compilation process

One could now manually add generate classes to the project. It has significant drawback: every time xsd is modified someone must:

- manually run xsd.exe
- manually update classes in project.

We will use better, more automatic approach:

1. add xsd file to project,
2. configure porject build tool, so classes will be generated automatically.

Project configuration:

1. Add your xsd file to the Class Library project.
2. Close solution i Visual Studio (File -> Close Solution). This step is required as Visual Studio does not support updating project file while it is open.
3. Open project definition file – .csproj file from library containing our class library project. Use File-> Open file to open it as a regular file.

The file has also XML syntax. We add two new Target rules[1] presented below. The second target rule describes how to generate .cs files from .xsd file by running given commands. The first target depends on the second one (it means that second one must be build successfully before running this target). The first target does not produce any output files. Instead it specifies that the generated files are added to list of files to be compiled. This way files are compiled during project build, bu they are not visible in project explorer. This is good behavior as otherwise one could modify the files, and the changes would be overwritten by a class generation during build.

Beside targets there are two additional rules: PropertyGroup that specifies that our target must be build before actual compilation and Generator in ItemGroup that specifies that classes should be re-generated when .xsd file is changed.

Note:

- Change name of xsd file to name you are using
- `&quot;` is an xml character entity – a method of escaping special characters in XML (c.f. section 4.6 of XML sepcification)

```
<!-- In .net older than 5.0 this element should exist somewhere in csproj. Add Generator rule to it.
  In newer dotnet you may need to add the whole element. -->
<ItemGroup>
  <ResourceFile Include="Zadanie1.xsd">
    <SubType>Designer</SubType>
    <Generator>MSBuild:UpdateClassesFromXSD</Generator>
  </ResourceFile>
</ItemGroup>

<!-- Append the following elements at the end of file, just before  </Project> -->
<Target Name="UpdateClassesFromXSD" DependsOnTargets="_UpdateClassesFromXSD" Condition="'@(ResourceFile)' != ''">
  <ItemGroup>
```

```
    <Compile Include="$(IntermediateOutputPath)fromxsd/*.cs" />
  </ItemGroup>
</Target>
<Target Name="_UpdateClassesFromXSD" Inputs="Zadanie1.xsd" Outputs="$(IntermediateOutputPath)fromxsd/*.cs">
  <Exec Command="mkdir $(IntermediateOutputPath)fromxsd" IgnoreExitCode="true" />
  <Exec Command="&quot;C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\xsd.exe&quot; /c /n:Library /out:$(IntermediateOutput
</Target>
<PropertyGroup>
  <CoreCompileDependsOn>UpdateClassesFromXSD;$(CoreCompileDependsOn)</CoreCompileDependsOn>
</PropertyGroup>
```

/If your xsd is not properly added as a resource file, the condition in the first target may be false, stopping classes from being build. Try removing `Condition` attribute from target./

After saving changes open the solution again and build class library porject. When you open project directory in file explorer you should see a directory `fromxsd` in obj subdirectory, containing generated classes.

MSBuild manual: https://msdn.microsoft.com/en-us/library/5dy88c2e.aspx

1.4 C# Program

In Class library project: create a static class `LibraryReader` which has one method `ReadLibrary` which has one argument – path to XML file and returns object containing deserialized document. Use XmlSerializer class from C# standard library.

In Console application:

1. Use `ReadLibrary` method to read XML file
2. Print surnames of all authors on standard output.

1.4.1 Why separation into two projects?

- we can easily define define several programs (e.g. console and gui) using the same core classses,
- we can divide responsibilities between teams of programmers.

2 Lab 2

During today's class you will:

- define more specific types for xml values in XSD,
- use keys and references to remove redundant data from XML,
- express key-value references in XSD,
- merge common elements of two types into a common supertype or common attribute group,
- verify an XML against an XSD in C# program.

2.1 More specific types

Add new requirements to the solution from previous week:

- book title consists of at least three characters,
- the only allowed values for language are: *EN*, *PL* i *TLH*,
- year must be at last 1120.

Such restrictions can be achieved by use of `restriction` element:

```
<xs:simpleType name="normalizedValue">
 <xs:restriction base="xs:double">
  <xs:minInclusive value="0"/>
  <xs:maxInclusive value="1"/>
 </xs:restriction>
</xs:simeplType>
```

`base` attribute specifies a type to restrict. Inside restriction element we add child elements expressing restrictions.

Common restrictions:

- for string

  pattern
  : regular expression
  min/maxLength
  enumeration
  : enumerate all possible values

- numeric types:

  min/maxInclusive
  min/maxExclusive

1. Define aforementioned restrictions in XML Schema file for your XML format,
2. check i Visual Studio reports incorrect values in XML document (it should report),
3. check if C# program written last week is able to read XML with values not confirming to restrictions (it should read such XML).

2.1.1 Verifying XML in C# program.

XML API in C# does not allow to simply deserialize XML and validate it against schema in single pass. To perform validation we need another reading pass with different kind of reader:

```
private static void ValidationHandler(Object sender, ValidationEventArgs args)
{
    if (args.Severity == XmlSeverityType.Warning)
        Console.WriteLine("Warning: {0}", args.Message);
    else
        Console.WriteLine("Error: {0}", args.Message);
}

//...
XmlReaderSettings settings = new XmlReaderSettings();
// Validator settings
settings.ValidationType = ValidationType.Schema;
settings.ValidationFlags |= XmlSchemaValidationFlags.ProcessInlineSchema;
settings.ValidationFlags |= XmlSchemaValidationFlags.ReportValidationWarnings;

// Here we add xsd files to namespaces we want to validate
// (It's like XML -> Schemas setting in Visual Studio)
settings.Schemas.Add("http://www.example.org/bookstore", "bookstore.xsd");

// Processing XSI Schema Location attribute
// (Disabled by default as it is a security risk).
settings.ValidationFlags |= XmlSchemaValidationFlags.ProcessSchemaLocation;

// A function delegate that will be called when
// validation error or warning occurs
settings.ValidationEventHandler += ValidationHandler;

XmlReader reader = XmlReader.Create("file.xml", settings);

// Read method reads next element or attribute from the document
// It will call ValidationEventHandler if some invalid
// part occurs
while(reader.Read()) {
}
```

The above code will print validation errors on standard output not giving any way to check if errors occured from within the program. You can add a boolean flag variable to store such information.

XmlReader on msdn: https://msdn.microsoft.com/en-us/library/system.xml.xmlreader.read

Modify the program in a way that XML will be deserialized and authors' names will be printed only if XML validates against XSD.

2.2 Removal of redundant author data by means of references

If our library contains several books by the same authors, each of books contains the same author data:

```
<library xmlns="http://example.org/jk/library">
  <book language="EN">
    <title>Red Prophet</title><year>1988</year>
    <author><name>Orson</name><name>Scott</name>
    <lastName>Card</lastName></author>
  </book>
  <book language="EN">
    <title>The Memory of Earth</title><year>1992</year>
    <author><name>Orson</name><name>Scott</name>
    <lastName>Card</lastName></author>
  </book>
  <book language="EN">
    <title>Earthborn</title><year>1995</year>
    <author><name>Orson</name><name>Scott</name>
    <lastName>Card</lastName></author>
  </book>
  <book language="EN">
    <title>A Discipline of Programming</title><year>1976</year>
    <author><name>Edsger</name>
    <lastName>Dijkstra</lastName></author>
  </book>
</library>
```

We can avoid such redundancy by moving authors to separate part of the document and assigning key (identifier) for each author. Simultaneously we remove author information from book entry and replace it with reference to author's identifier.

```
<library xmlns="http://example.org/jk/library">
  <book language="EN">
    <title>Red Prophet</title><year>1988</year>
    <author ref="osc"/>
  </book>
  <book language="EN">
    <title>The Memory of Earth</title><year>1992</year>
    <author ref="osc"/>
  </book>
  <book language="EN">
    <title>Earthborn</title><year>1995</year>
    <author ref="osc"/>
```

```
  </book>
  <book language="EN">
    <title>A Discipline of Programming</title><year>1976</year>
    <author ref="ed"/>
  </book>
  <author id="osc"><name>Orson</name><name>Scott</name>
  <lastName>Card</lastName></author>
  <author id="ed"><name>Edsger</name>
  <lastName>Dijkstra</lastName></author>
</library>
```

Change structure of your document to avoid redundant author information. Change XSD to mirror new structure of the document. /Note: It is perfectly valid that we have two kinds of elements of name author that have different type as long as they are in different parts of the document. Note2: Use plain type like string or int for idetifiers. Never use ID and IDREF types!! Those types are highly deprecated./

Once xsd is consistent with new document structure we have to enforce only valid references in the document. In current version we can define author reference that points to nonexistent author.

Keys
: define that some value is an unique identifier of some entity. No two entities with the same key may exist in the document.

References
: define that some property of an entity must refer do a existing key value. If no corresponding key exists then the reference is invalid.

We modify our xsd by adding keys (key element) and references (keyref element). Those definitions must be put in some common ancestor of elements that refer to and elements that are referred by. Usually we can use document root for that. We put key and keyref elements after type definition in root element:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://example.org/jk/library"
    elementFormDefault="qualified"
    xmlns:lib="http://example.org/jk/library"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
 <xs:element name="library">
  <complexType>
    ....
  </complexType>
  <xs:key name="authorKey">
   <xs:selector xpath="lib:author"/>
   <xs:field xpath="@id"/>
  </xs:key>
  <xs:keyref name="authorRef" refer="lib:authorKey">
   <xs:selector xpath="lib:book/lib:author"/>
   <xs:field xpath="@ref"/>
  </xs:keyref>
 </xs:element>
</xs:schema>
```

Key definition consists of name:

name
: unique name that will be used in reference definition

selector
: entity to be referred

field
: field (key) which will be matched by references

field must point to a `simpleType` value. It may occur more than once – this way we will define compound key.

Reference definition consists of:

name
: unique reference identifier (usually not used anywhere)

refer
: name of a key reference refers to

selector
: element that contains a reference

field
: field that has to be equal to value in key (foreign key)

Number of occurrences of `field` in `key` and `keyref` must be equal.

`selector` and `field` have attribute `xpath` – an expression choosing elements from xml document tree in XPath language. Very short introduction to XPath:

- XPath resembles file path
- Selects a **set** of elements by matching ancestor names: `a:el1/a:el2` selects set of all `el2` that are direct descendants of `el1` such that are direct descendant of current element. The paths are relative to current element which is:
  - an element in which key(ref) definition is contained in case of selector,
  - element selected by the selector in case of field.
- if path segment begins with `@` it means attribute name (without at sign it is element name)

Be careful about namespaces:

- XPath expressions must contain **prefixed** names of element. It will not work if we use namespace mappend without a prefix.
- key names are defined in target namespace. When defining a keyref you must use namespace prefix in reference to key name.
- attribute names (as long as you are not using `attributeFormDefault="qualified"`), are written without a prefix.

1. Modify XSD to enforce correct key-value references.
2. Try entering a bad reference to the XML. Where is the error reported by Visual Studio? Why is it that late in document?
3. Check if C# program allows malformed references when reading the file.

2.3 Common parts of defined types

Extend the document format to store new type of entity: a journal.

Journal properties:

- editors (the same type like authors in books),
- year (like book),
- title (like book),
- issue number (integer).

Obviously both journal and book have several properties in common. It is possible to repeat some XSD code and write it as separate types, but better approach is not to repeat any parts.

Now we consider two methods of writing the common part of a type.

2.3.1 Type extension

One possibility is to extend previously defined type. It is similar to class inheritance in OOP:

- create a type A which contains common part of two types,
- define types B and C that extend A.

Example extension:

```
<xs:schema targetNamespace="http://example.org/jk/library"
    elementFormDefault="qualified"
    xmlns:lib="http://example.org/jk/library"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
<xs:complexType name="A">
  <xs:sequence>
    <xs:element name="X" type="xs:integer"/>
  </xs:sequence>
  <xs:attribute name="Y" type="xs:string"/>
</xs:complexType>

<xs:complexType name="B">
  <xs:complexContent>
    <xs:extension base="lib:A">
      <xs:sequence>
        <xs:element name="Z" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="Q" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

`complexContent` can contain eiter

extension
: extending another type or

restriction
: restricting some properties of existing type.

In typical cases we use `extension`. Note that this is not OOP and values of extended type may not appear in XML in places where base type is expected.

Employing type extension add journals to your XML Schema file.

2.3.2 complexType and complex content

It is worth to know that complexType containing sequence (or another element definitnion) is short syntax for defining restriction on `anyType`:

```
<xs:complexType>
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:element name="Z" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="Q" type="xs:string"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

2.3.3 Element and attribute groups

Another possibility is to define named groups of attributes and elements.

Element and attribute groups are named sequences of elements `element` and `attribute` respectively. They can be referenced in places where we usually put elements (attributes) directly. In will be equivalent to copying and pasting the group contents to the place.

```
<xs:attributeGroup name="names">
 <xs:attribute name="firstName" type="xs:string"/>
 <xs:attribute name="middleName" type="xs:string"/>
 <xs:attribute name="lastName" type="xs:string"/>
</xs:attributeGroup>

<xs:group name="XYZ">
 <xs:sequence>
   <xs:element name="X" type="xs:string"/>
   <xs:element name="YZ">
    <complexType>
     ...
    </complexType>
   </xs:element>
  </xs:sequence>
</xs:group>

<xs:complexType name="T">
 <xs:sequence>
  <xs:element name="A" type="xs:string"/>
  <xs:group ref="lib:XYZ" />
 </xs:sequence>
 <xs:attributeGroup ref="lib:XYZ"/>
 <xs:attribute .../>
 <xs:attribute .../>
 <xs:attributeGroup ref="lib:ASDF"/>
</xs:complexType>
```

groups vs. type extension:

- a complexType can extend only one type, but it can contain any number of attribute and element group references
- extension ``inherits" both attributes and elements, groups for elements and attributes are separable.

Prepare an alternative XML Schema file that uses groups instead of extension.

2.3.4 groups and extension in xsd.exe generated classes

Generate C# classes from both xsd file variants. Do they differ in any way?

2.3.5 Extending C# program

Source files generated by xsd tool contain classes that are marked as partial. It allows continuation of such class implementation on another (not auto-generated) source file. To continue implementation of partial class in another file you need to create a class that:

- has the same name,
- is in the same namespace,
- has partial modifier.

Both parts of class implementation will be merged during compilation.

More about partial classes: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods

1. Using partial add `ToString` implementations to all generated classes.
2. Modify the program to print whole deserialized file, not only authors' names.
3. Print title of the oldest book in the library.

3 Lab 3

For 10 points. You should know how to:

- design XML structure for given data set,
- express this structure in XSD file:
  - correctly reference namespaces,
  - define specific type restrictions,
  - define key-value references,
  - reasonably handle common parts of types,
- generate C# code from XSD
- hook up automatic code generation with C# project
- deserialize XML to generated classes
- verify XML against XSD in C# program.

Footnotes:

[1] Those targets are conteptually similar to targets you know from unix makefiles. The define how to transform input files into output files.

Author: Jan Karwowski
Created: 2021-02-24 śro 08:27
Validate