POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

**DEPARTAMENTO DE ENGENHARIA INFORMÁTICA**

www.dei.estg.ipleiria.pt

# Systems Integration

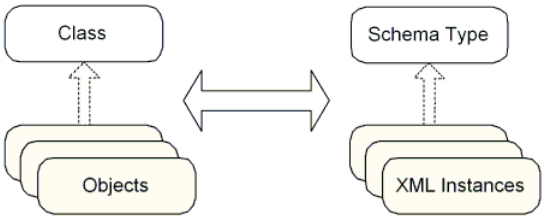
## Extensible Markup Language – XML

**Goals/Topics:**

– Understand the concepts related to XML and its importance in the current information systems
– Identify the main components about the manipulation of information in XML format
– Understand XML Schema definition language (XSD)
– Create a XML Schema (XSD)
– Handle XML though the .NET Framework
– Manipulate XML through the Microsoft implementation of DOM (`XmlDocument`)

**Duration: 1,5 classes**






**OO vs XML concepts**

# – PART 1 –

# XML STRUCTURE AND XML SCHEMAS

Based on the XML example shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book category="CHILDREN">
     <title>Charlottes Web</title>
    <author>E.B. White</author>
    <year>1952</year>
    <price>12.60</price>
  </book>
  <book category="WEB">
    <title>Beginning XML</title>
    <author>Joe Fawcett</author>
    <year>2012</year>
    <price>31.30</price>
    <rate>4</rate>
  </book>
  <book category="WEB">
    <title>Programming .NET Web Services</title>
    <author>Alex Ferrara</author>
    <year>2002</year>
    <price>38.36</price>
    <rate>3</rate>
  </book>
</bookstore>
```

**Figure 1 – Example of a XML document for a *bookstore***

1. Identify the existing elements and attributes.

2. Which changes would you make to the XML structure to:

   2.1. Identify the book *publisher*.

   2.2. Allow a book to have several *authors*.

## What is a XML Schema (.xsd file)?

The **XML Schema definition language (XSD**) provides a type system for XML processing environments. An XML Schema makes it possible to describe types that you intend to use. An XML document that conforms to an XML Schema type is often referred to as an instance document, very much like the traditional object-oriented (OO) relationship between classes and objects. The schema file also uses the XML notation.

The **<schema>** element is the root of the **XML Schema file (XSD file)**:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema ...>
...
...
</xs:schema>
```

Syntax Example for the elements and attributes inside the XSD file:

```
<xs:element name="name" type="xs:string"/>

<xs:attribute name="lag" type="xs:string"/>
```

**Complex types example: c**onsidering this XML example.

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

**Alternative 1:**

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

**Note:** Each of these elements may contain attributes as well.

**Alternative 2:**

```
<xs:element name="employee"
type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

In this example, the "employee" element can have a type attribute that refers to the name of the complex type to use.
The advantage is that several elements can refer to the same complex type.

**Figure 2 – Example of an XML Schema's content**

**3.** The aim is to generate a **XML schema** (.XSD file) based on a XML file (.XML file). Thus,

3.1. Use Visual Studio to create a new XML file and write into the file the text shown in Figure 1.

3.2. Then select XML » Create Schema to generate the corresponding Schema. A Schema file (.XSD) will be automatically generated. The generated file is identical to *Alternative 1*.

3.3. Identify how are defined the elements and attributes.

3.4. Look at the auto-generated file. What is a complex type?

3.5. What is the difference between `<sequence>`, `<choice>` and `<all>`?

3.6. Where are the elements' details changed? Namely:

- The number of *minimum occurrences* and *maximum occurrences*;
- The element type.

3.7. Change the generated Schema file structure to look like the ***Alternative 2*** (give names to the `complexType` allowing to reuse them).

3.8. Change the Schema file considering that, from now one, the book can accept a **bookformat** element and an **isbn** attribute.

3.9. Add the following restrictions in the schema file:

3.9.1. The **category** attribute accepts the following values: WEB, CHILDREN, SCIENCE, ROMANCE, etc.

3.9.2. The **rate** element accepts only values between 1 and 5.

3.9.3. The **year** element only accepts 4 digits.

3.9.4. The **bookformat** element can have the following values: paperback, electronic book text, hardback, ebook, loose-leaf.

**4.** Auto-generate a new XML file based on the changed XML Schema – XSD file. The aim is to test the changes implemented in the new XML Schema File.

4.1. Right click the root element in the **XML Schema Explorer** and select **Generate Sample XML** option. Fill the file with *dummy data* to test if all the restrictions were applied.

4.2. Using the Visual Studio IDE try to see if the XML file is well formed and valid. Tip: use the *Properties* tool (F4 or menu option View) in VS.

## – PART 2 –

## MANIPULATING XML USING DOM AND XPATH

Create a new project of type *Windows Form Application* called **ProjectXML.** All exercises must be solved using C# language. (Instead, you may also download the base project available in the course web page).

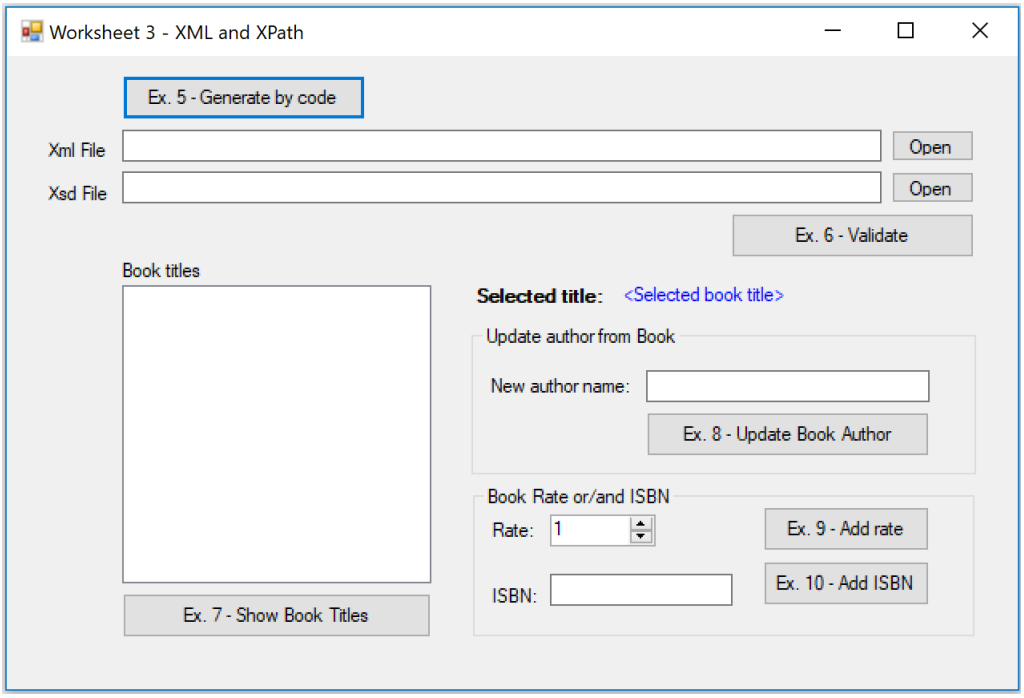



**Figure 3 – Interface for the project being created.**

**5.** Implement a function that creates a new XML file like the one depicted in Figure 1. Use the DOM classes (use the `System.Xml` namespace) to create by code the elements and attributes required by the file structure in Figure 1. Figure 3 shows a "small sample" of how to create an XML file like the one depicted in Figure 1. To test the implemented function, add a button to the Form and use it to call the function.

**6.** Implement a function to programmatically verify if the XML file is **well formed and valid**. (Use the classes: `Xml.XmlDocument` and `ValidationEventHandler`). Send validation failures to the console.

*Tip*: explore the characteristics of the `XmlDocument` class in the help (MSDN)

7. Implement a function that returns all the book ***titles*** in the XML file. The book titles can be shown in the Form in a *ListBox* component.

8. Implement a function to update the book **author** of a given book (by its title). Save the changes made in the XML file. Test the function allowing the user to: select the book title (e.g., from the previous *Listbox*) and define the new author name by using a *textbox*.

9. Implement a function to add the **rate** element to a book. The rate element must be added to a specific book depicted by its title. Save the changes made in the XML file.

10. Implement a function to add the **isbn** attribute to a book. The isbn attribute must be added to a specific book depicted by its title. Save the changes made in the XML file.

DOM (*Document Object Model*): Example showing how to create the XML file in memory using DOM:

```
        XmlDocument doc = new XmlDocument();

        // Create the XML Declaration, and append it to XML document
        XmlDeclaration dec = doc.CreateXmlDeclaration("1.0", null, null);
        doc.AppendChild(dec);

        // Create the root element
        XmlElement root = doc.CreateElement("bookstore");
        doc.AppendChild(root);
        // Create Books
        // Note that to set the text inside the element,
        // you use .InnerText
        // You use SetAttribute to set attribute
        XmlElement book = doc.CreateElement("book");
        book.SetAttribute("category", "CHILDREN");
        XmlElement title = doc.CreateElement("title");
        title.InnerText = "Chalottes Web";
        XmlElement author = doc.CreateElement("author");
        author.InnerText = "E.B. White";
             ...
    book.AppendChild(title);
    book.AppendChild(author);
    root.AppendChild(book);
         ...

    string xmlOutput = doc.OuterXml;
    Console.WriteLine(xmlOutput);

    // OR save the XML in the disc to test in the next exercise
     doc.Save(@"example.xml");
```

and the output is as follows:

```
<?xml version="1.0"?>
<bookstore>
  <book category="CHILDREN">
    <title>Charlottes Web</title>
    <author>E.B. White</author>
  </book>
</bookstore>
```

**Figure 4 – Example code using .NET's DOM functionalities to create a XML file**

## [ADDITIONAL EXERCISES]

**11.** Add a button to the Form that shows the number of books from a specific CATEGORY. Implement the required behaviour as a function that receives as argument the category name and returns the number of books.

**12.** Figure 4 shows an XPath editor application to test XPath expressions. Create a *Windows Forms Application* project with the same interface. It must allow writing an XPath expression and the XML file path. The results must be presented when the user clicks the "Execute" button.

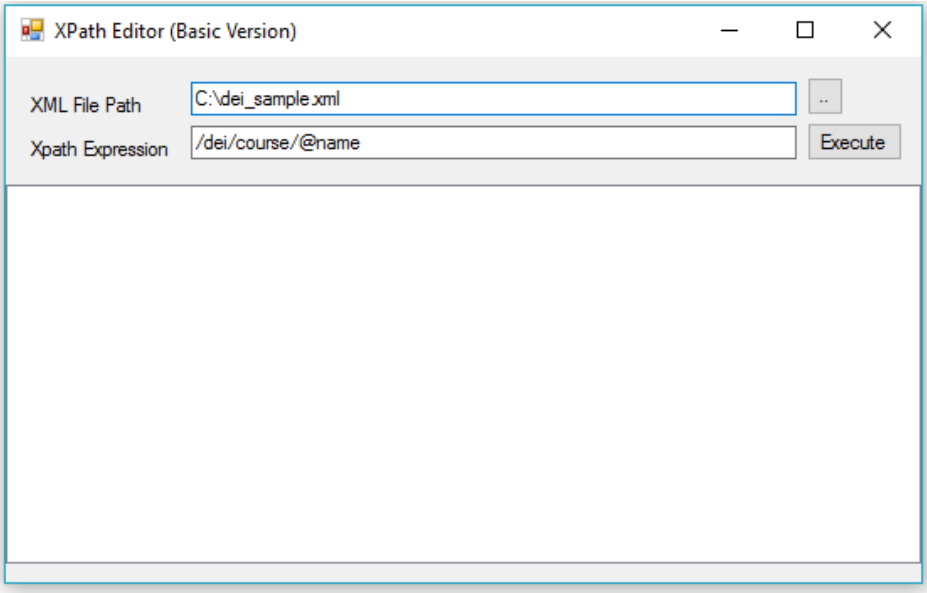



**Figure 5 – Interface example of an XPath editor**

**13.** The Department of Informatics Engineering stores, in XML format, information about the courses available in each semester. Download the two sample files from the course web page, named `dei_sample.xml`.

Based on the content of the XML files create the following **XPath** expressions:

13.1. Go to the first course from the XML file; /dei/course[1]

13.2. Get the name of the second course; /dei/course[2]/@name DAD

13.3. Get the name of the last course; /dei/course[last()]/@name programação de Jogos

13.4. Get the name of the second to last course; /dei/course[last()-1]/@name

13.5. Get the name of the first teacher of the first course; /dei/course[1]/teachers/teacher[1]/name

13.6. Get the teachers names that teach the IS course; /dei/course[@name="Integracao de Sistemas"]/ teachers/teacher/name

13.7. Get the names of the courses;/dei/course/@name

13.8. Get the student number whose name contains "Manuel"; /dei/course/students/student[name='Manuel']/number

13.9. Get the student number whose name starts with "M"; /dei/course/students/student[starts-with(name, 'M')]/number

13.10. Get the course regime from the courses whose name starts with "I" and have 6 ECTS; /dei/course[starts-with(@name, 'I') and @ects=6]/scheme

13.11. Get the names of all the courses associated with the teacher that has the name "Francisco"; /dei/course[teachers/teacher/name='Francisco']/@name /dei/course/teachers/teacher[name="Francisco"]/parent::node()/parent::node()/@name

13.12. Get the ECTS of all courses. /dei/course/@ects