

XML, DTD, XSD y XSL

Conceptos generales

- **XML** (*eXtensible Markup Language*). Metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes.
- **DTD** (*Document Type Definition*). Es un documento que define la estructura de un documento XML: elementos, estructura, entidades, notaciones, etc., orden en que pueden aparecer, número de veces que pueden aparecer, cuáles pueden ser hijos de cuales, etc. El procesador XML utiliza el DTD para verificar si un documento XML es válido.
 - El DTD se especifica a través de la etiqueta **DOCTYPE**. El DTD puede estar incluido en el propio documento XML, ser un documento externo o una combinación de ambos métodos.
- **XSD** (*XML Schema Definition*). Al igual que el DTD, el XSD también permite definir la estructura de un documento XML.
- **XSL** (*eXtensible Stylesheet Language*). Es un lenguaje que permite dar estilo y formato a documentos XML, transformándolos un documento XML en otro formato como por ejemplo un documento HTML capaz de presentarse a través de un navegador.
- **XPath**. XPath permite formar expresiones que se pueden utilizar en un XSL con el objetivo de poder navegar en el documento XML y poder acceder a elementos, subelementos, atributos, filtrar datos, etc.

Declaración XML

Cabecera XML

Simplemente, indicamos la etiqueta que permite definir la cabecera del documento XML, indicando la versión y la codificación.

```
<?xml version="1.0" encoding="UTF-8">
```

Standalone

Podemos añadir un parámetro para especificar que el documento debe ser “**standalone**”; es decir, autónomo (será un documento que no haga referencia a otros documentos). Indicará, por ejemplo, que no se pueden utilizar DTD externos (y, si se hace, dará un error).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

Cheat Sheet XML, DTD, XSD y XSL

XML	factures.xml <pre><?xml version="1.0" encoding="UTF-8" standalone="yes" ?></pre>
DTD	factures.xml <pre><?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE factures SYSTEM "factures.dtd" > <factures> ... </factures></pre>
	factures.dtd <pre><!ELEMENT factures (factura*)> ...</pre>
XSD	factures.xml <pre><factures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="factures.xsd" > ... </factures></pre>
	factures.xsd <pre><?xml version="1.0" encoding="UTF-8"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"> <xs:element name="factures"> ...</pre>
XSL	factures.xml <pre><?xml version="1.0" encoding="UTF-8" ?> <?xml-stylesheet type="text/xsl" href="factures.xsl"?></pre>
	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:exsl="http://exslt.org/common" extension-element-prefixes="exsl"> <xsl:template match="/"> ...</pre>

DTD

Declaración del DTD

En el propio documento XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE nombre [
    ... declaraciones ...
]>
```

- **nombre:** el nombre del tipo de documento XML. Debe coincidir con el nombre del elemento raíz del documento XML.

Documento externo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE nombre SYSTEM "uri">
```

- **SYSTEM:** nos indica que es un DTD que no es público (lo tenemos localizado en la máquina local, en un servidor privado, etc). El archivo DTD especificado en la uri tendría extensión ".dtd", por ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE agenda SYSTEM "agenda.dtd">
<agenda>
    ...
</agenda>
```

Documento externo + interno

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE nombre SYSTEM "uri" [
    ... declaraciones ...
]>
```

Si queremos indicar que es un DTD público, lo hacemos de la siguiente forma:

```
<!DOCTYPE nombre PUBLIC "fpi" "uri">
```

- **fpi:** es el Identificador Público Formal del DTD contra el que queremos validar el XML. Por ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<!NOTATION png SYSTEM "image/png">
]>

<autores>
  <autor>
    <nombre>Autor: &nomAutor;&copyright;2020</nombre>
    <imagen src="companyLogo" />
  </autor>
  &autor1;
</autores>
```

En el ejemplo anterior, se incluye el contenido del fichero “**autor_externo.xml**” justo en el punto donde se coloca la sentencia `&autor1;`

Por alguna razón, esto no funciona al abrir el XML en un navegador. Si funciona haciendo uso de la herramienta `xmllint` (explicada después), pero debemos proporcionar el parámetro `--noent` para que procese y resuelva todas las entidades declaradas en el DTD y utilizadas en el XML.

Validación de un XML con su DTD

La herramienta **xmllint** permite validar un XML con la línea de comandos:

```
xmllint --valid --noout --noent documento.xml
```

- **--valid**: indica que queremos validar el XML contra el DTD que tenga especificado.
- **--noout**: si el XML es válido no mostrará ninguna salida en pantalla (si no se indica, mostrará el XML procesado en pantalla).
- **documento.xml**: el nombre del XML que queremos validar
- **--noent**: permite indicar que queremos resolver y procesar todas las entidades declaradas en el DTD y utilizadas en el XML. Lo lógico es que si utilizamos esta opción quitemos la opción `--noout` para poder ver el resultado final del XML procesado.

JAXB

JAXB (Java Architecture for XML Binding) es una librería que permite trabajar con ficheros XML en Java y que viene instalada dentro del paquete **javax.xml**. Esta librería permite parsear ficheros XML y construir nuevos haciendo uso de la potencia de las clases y objetos y las anotaciones.

Construir jerarquía de clases desde el DTD

Gracias a la herramienta de terminal `xjc` (Java XML Binding Compiler) podemos construir toda la jerarquía de clases con las anotaciones correspondientes a través de la definición de un DTD.

```
xjc -dtd -d agenda_basica_java -p
com.davidlopezcastellote.agenda.xml agenda_basica.dtd
```

- `-dtd`: crear la estructura de clases a partir de un DTD
- `-d <directorio>`: la carpeta donde se guardará la estructura de clases
- `-p <paquete>`: el paquete Java donde se colocarán las clases
- `<dtd>`: por último, se debe especificar el fichero DTD con la definición

```
<!-- DTD -->
<!ELEMENT agenda (contacto)*>
  <!ELEMENT contacto (nombre_completo, edad?)>
    <!ELEMENT nombre_completo (#PCDATA)>
    <!ELEMENT edad (#PCDATA)>
```

```
MacBook-Pro-de-David-5:agenda david$ xjc -dtd -d agenda_basica_java -p com.davidlopezcastellote.agenda.xml agenda_basica.dtd
Analizando un esquema...
Compilando un esquema...
com/davidlopezcastellote/agenda/xml/Agenda.java
com/davidlopezcastellote/agenda/xml/Contacto.java
com/davidlopezcastellote/agenda/xml/ObjectFactory.java
MacBook-Pro-de-David-5:agenda david$
```

Agenda.java

```
Agenda.java  Contacto.java  ObjectFactory.java  Main.java
8
9 package com.davidlopezcastellote.agenda.xml;
10
11 import java.util.ArrayList;
12 import java.util.List;
13 import javax.xml.bind.annotation.XmlAccessType;
14 import javax.xml.bind.annotation.XmlAccessorType;
15 import javax.xml.bind.annotation.XmlRootElement;
16 import javax.xml.bind.annotation.XmlType;
17
18
19 @XmlAccessorType(XmlAccessType.FIELD)
20 @XmlType(name = "", propOrder = {
21     "contacto"
22 })
23 @XmlRootElement(name = "agenda")
24 public class Agenda {
25
26     protected List<Contacto> contacto;
27
28     public List<Contacto> getContacto() {
29         if (contacto == null) {
30             contacto = new ArrayList<Contacto>();
31         }
32         return this.contacto;
33     }
34
35 }
```

```

<div class="factura">
  <h2>Factura <xsl:value-of select="./@num-factura" /></h2>
  // ...
</div>
</xsl:template>

```

Bucles (xsl:for-each)

Permiten mostrar una serie de etiquetas HTML (el output) para cada elemento encontrado en la expresión de XPath.

```

<xsl:for-each select="./compra/producte">
  <tr> ... </tr>
</xsl:for-each>

```

xsl:value-of

Permite mostrar en el HTML el valor de una variable o una expresión de XPath.

Siendo \$codi una variable:

```

<td><xsl:value-of select="$codi" /></td>

```

```

<xsl:template match="celler/factures/factura">
  <h2>Factura <xsl:value-of select="./@num-factura" /></h2>
</xsl:template>

```

Variables (xsl:variable)

Permiten definir una variable que después podrá ser utilizada en otras etiquetas XSL. El valor de la variable podrá ser un literal (ej. un número constante), una expresión de XPath, una operación entre otras variables...

```

<xsl:for-each select="./compra/producte">
  <tr>
    <xsl:variable name="unitats" select="." />
    <xsl:variable name="codProd" select="./@cod-producte" />
    <xsl:variable name="preu"
      select="/celler/productes/vi[@codi=$codProd]/@preu" />
    <xsl:variable name="preuTotal" select="$unitats*$preu" />
    <td><xsl:value-of select="$codProd" /></td>
    <td><xsl:value-of select="$preu" />€</td>
    <td><xsl:value-of select="$unitats" /></td>
  </tr>
</xsl:for-each>

```