

# DIPLOMADO

## Desarrollo de Sistemas con Tecnología Java



Módulo 5. Uso de XML en la creación  
de interfaces web e intercambio de información

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

[tic.unam.mx](http://tic.unam.mx)



DGTIC

# DIPLOMADO

## Desarrollo de Sistemas con Tecnología Java

Módulo 5. Uso de XML en la creación  
de interfaces web e intercambio de información

Autor: Jorge Alberto Montalvo Olvera



## **UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

### **Rector**

José Narro Robles

### **Secretario General**

Eduardo Bárzana García

### **Director General de Cómputo y de Tecnologías de Información y Comunicación**

Felipe Bracho Carpizo

### **Directora de Docencia en Tecnologías de Información y Comunicación**

Adela Castillejos Salazar

## **Diplomado Desarrollo de Sistemas con Tecnología Java**

### **Módulo 5: Uso de XML en la creación de interfaces web e intercambio de información**

#### **Coordinación de la publicación**

María Guadalupe Izquierdo Dyrzo

#### **Autor**

Jorge Alberto Montalvo Olvera

#### **Revisión técnica**

Nidia Cendejas Cervantes

#### **Diseño editorial**

Lidia Angelina Castillo Peña

#### **Editor**

DR © Universidad Nacional Autónoma de México  
Ciudad Universitaria, Coyoacán, CP. 04510, México, DF.

Todos los nombres propios de programas, sistemas operativos, equipos, hardware, etc. que aparecen en estas notas son marcas registradas de sus respectivas compañías u organizaciones.

### **DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN**

Circuito exterior s/n, Ciudad Universitaria, Coyoacán, CP. 04510, México DF.

ISBN en trámite

Prohibida la reproducción total o parcial por cualquier medio sin la autorización escrita del titular de los derechos patrimoniales

1ª. Edición, Junio de 2014

Impreso y hecho en México

# CONTENIDO

<b>CONTENIDO.....</b>	<b>III</b>
<b>1. ORIGEN DEL XML .....</b>	<b>5</b>
1.1. Definición de XML .....	6
1.2. Estructuración y sintaxis.....	11
1.3. Estructuración de contenidos XML.....	14
<b>2. FUNDAMENTOS DE XML .....</b>	<b>17</b>
2.1. Creación de un archivo XML .....	17
2.2. Creación de archivos DTD (Document Type Definition).....	17
2.3. XML Schema .....	25
<b>3. CONSTRUYENDO DOCUMENTOS VÁLIDOS.....</b>	<b>27</b>
3.1. Validación .....	27
<b>4. DESPLIEGUE DE LA INFORMACIÓN .....</b>	<b>32</b>
4.1. CSS .....	32
4.2. XSL .....	35
4.3. XSLT.....	36
4.4. XPATH .....	39
<b>5. MODELOS DOM.....</b>	<b>42</b>
5.1. Nodos.....	43
5.2. Elementos DOM.....	44
5.3. Trabajando con Nodos .....	44
<b>6. JAVA Y XML .....</b>	<b>46</b>
6.1. JDOM .....	46
6.2. JAXP .....	49
6.3. JAX-RPC.....	50
<b>MESOGRAFÍA.....</b>	<b>54</b>

# 1 ORIGEN DEL XML

De manera continua, los medios de comunicación que se especializan en tecnologías de la información toman mayor importancia dentro del lenguaje XML, al que definen como la lengua base de internet. Desde que el lenguaje XML hizo su aparición ha cobrado un importante protagonismo en el diseño de publicaciones web. Lo cual lo hace posicionarse como punto importante para el diseño de aplicaciones y servicios para la web.

El lenguaje XML es una evolución de los lenguajes SGML (Standard Generalized Markup Language) definido en 1986 y HTML (Hypertext Markup Language) creado en los años noventa.<sup>1</sup>

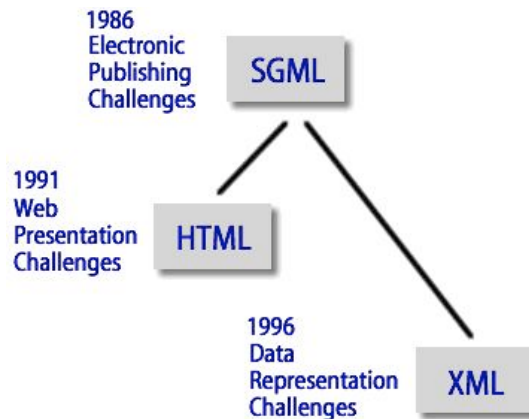


Figura 1. Evolución de XML, consultado el 25 febrero de 2014

SGML es la norma que ha tenido un mayor impacto en el mundo de la edición electrónica. Su origen se encuentra en el lenguaje de etiquetas GML (Generalized Markup Language). SGML parte del concepto de tipo de documento.

Un tipo de documento puede interpretarse como una abstracción de documentos que comparten un objetivo, propiedades y necesidades de tratamiento comunes.

Otro punto importante de SGML es la inclusión de etiquetas dentro del texto de los documentos para diferenciar su estructura y su contenido informativo.

A partir de este punto, SGML definió:

- La sintaxis que debía utilizarse para diseñar un conjunto de etiquetas aplicables a cada documento.
- La forma en la que se deben de intercalar etiquetas en el texto de un documento.

<sup>1</sup> <http://csharpcomputing.com/XMLTutorial/xmlrevolution.jpg>

SGML proporciona grandes ventajas pero la dificultad y los costos que implica el proceso de creación de documentos SGML y su alta curva de aprendizaje constituyeron las principales desventajas que debían afrontar las organizaciones interesadas en adoptar este formato lo cual frenó su adopción.

Por otro lado HTML es una aplicación del lenguaje SGML que especifica cómo se deben de codificar los documentos para distribución en el web (world wide web). HTML fue creado en los años noventa cuando la web se desarrolló con el objetivo, de que la comunidad científica tuviera la facilidad de comunicarse e intercambiar información a través de internet y así poder crear sistemas de información.

HTML de la misma manera que SGML era independiente de la plataforma hardware o de software, lo cual lo convertía en la solución perfecta para los problemas de intercambio de documentación en formato electrónico, pero pronto se hicieron manifiestas sus limitaciones, algunas de ellas:

- La incapacidad de presentar las características tipográficas y presentaciones complejas de los documentos.
- La falta de capacidad expresiva del lenguaje, debido a que solo se puede utilizar un número limitado de etiquetas predefinidas en su especificación.
- La evolución del lenguaje HTML estuvo condicionada por la Netscape y Microsoft.

Una de las alternativas para estas limitaciones fue la creación del lenguaje HTML Dinámico (DHTML) y el lenguaje XML.

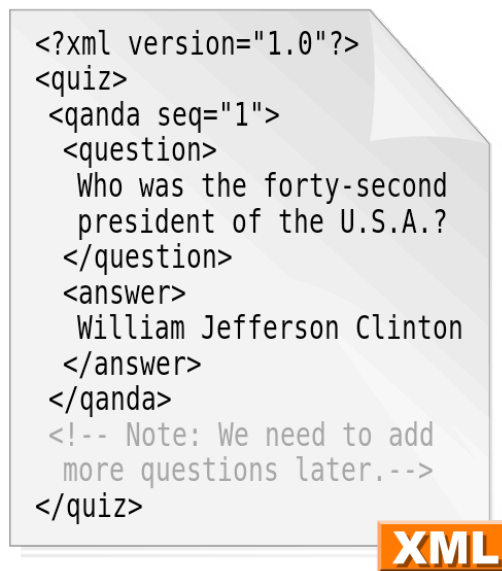
## 1.1. Definición de XML

El lenguaje XML (eXtensible Markup Language), es un lenguaje para representar datos e información. Es importante mencionar que no es un lenguaje de programación, lo que sí es posible es procesar documentos XML con programas realizados en cualquier lenguaje de programación.

El lenguaje XML fue creado por el World Wide Web Consortium (W3C) con el propósito de tener un lenguaje de etiquetas optimizado para poder ser utilizado en Internet. Parte de sus objetivos es combinar la simplicidad de HTML con la capacidad expresiva de SGML. El W3C es quien rige las especificaciones tanto de HTML como de XML. En el caso de XML iniciando con la especificación XML 1.0. La figura 2 muestra un documento XML.<sup>2</sup>

---

<sup>2</sup> <http://upload.wikimedia.org/wikipedia/commons/thumb/6/68/XML.svg/500px-XML.svg.png>



```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

*Figura 2. Ejemplo de un documento XML*

Una manera de facilitar la creación de documentos XML es el uso de DOM (Document Object Model) el cual ofrece un modelo de objetos que permiten acceder así como manipular las partes de un documento según un modelo orientado a objetos.

Las principales características de XML son las siguientes:

- Permite tener un conjunto de etiquetas abiertas y ampliables
- Diferencia entre estructura y presentación de los datos
- Gestiona de hipervínculos avanzada
- Modularidad en la información

Es importante aclarar que XML es un perfil de SGML y no una aplicación SGML, como en el caso de HTML. Así, frente al conjunto de etiquetas predefinidas que conforman HTML, el lenguaje XML ofrece la posibilidad de definir nuevas etiquetas para codificar la estructura y contenido de distintos tipos de documento.

Al igual que SGML, en XML es posible crear DTD (Document Type Definition) que indiquen las reglas que debe seguir cada tipo de documento.

Para indicar cómo se debe de presentar un documento en pantalla o en papel, se requiere crear una hoja de estilo aparte del documento XML y asociarla posteriormente al documento. Para la creación de hojas de estilo para documentos XML se tienen: CSS (Cascade Style Sheets) y XSL (Extensible Stylesheet Language).

Ejemplo de CSS aplicado a un documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="estilo.css"?>
<libro>
  <titulo>La vida esta en otra parte</titulo>
  <autor>Milan Kundera</autor>
  <fechaPublicacion anio="1973"/>
</libro>
```

Contenido de archivo estilo.css

```
titulo {
  color: red;
}
```

Resultado de la aplicación de la hoja de estilo CSS.

La vida está en otra parte Milan Kundera

Ejemplo de XLS aplicado a un documento XML:

Archivo de datos XML

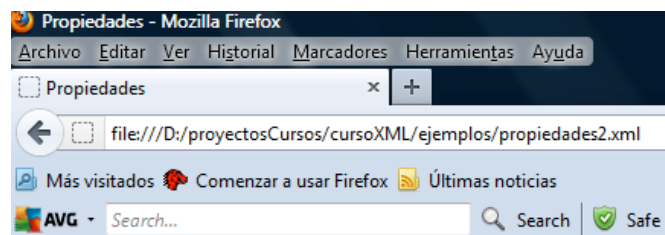


```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="propiedadesxsl.xsl"?>
<propiedades>
  <propiedad id="1">
    <direccion>Francisco I. Madero No. 234</direccion>
    <precio>$100,000</precio>
    <comentarios>Cerca del metro de linea 5</comentarios>
  </propiedad>
  <propiedad id="2">
    <direccion>Marte No. 15</direccion>
    <precio>$230,000</precio>
    <comentarios>Colonia muy tranquila</comentarios>
  </propiedad>
  <propiedad id="3">
    <direccion>Av. Tlahuac No. 4174</direccion>
    <precio>$150,000</precio>
    <comentarios>Buena vista de paisaje</comentarios>
  </propiedad>
  <propiedad id="4">
    <direccion>Misterios No. 21</direccion>
    <precio>$80,000</precio>
    <comentarios>Barrio conflictivo</comentarios>
  </propiedad>
  <propiedad id="5">
    <direccion>Pueblo Quieto No. 1000</direccion>
    <precio>$600,000</precio>
    <comentarios>Muy tranquilo</comentarios>
  </propiedad>
  <propiedad id="6">
    <direccion>Sin nombre</direccion>
    <precio>$300,000</precio>
    <comentarios>Sin comentarios</comentarios>
  </propiedad>
</propiedades>
```

## Hoja de estilo XLST

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
  <html>
  <head>
  <title>Propiedades</title>
  </head>
  <body>
  <h1>Propiedades Disponibles</h1>
  <xsl:for-each select="//propiedad">
    <xsl:if test="@id='1'">
      <h1><xsl:value-of select="direccion" /></h1>
      <h1>Es la mejor</h1>
    </xsl:if>
  </xsl:for-each>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Resultado de la aplicación de la hoja de estilo XSLT:



**Propiedades Disponibles**  
**Francisco I. Madero No. 234**  
**Es la mejor**

La edición de documentos XML persigue los siguientes objetivos:

- Hacer diferencia entre el contenido informativo y la estructura de los documentos de los documentos de su presentación.
- Hacer explícita la estructura y los contenidos informativos de los documentos mediante la utilización de etiquetas.
- Poder crear documentos portables que puedan intercambiarse y proporcionar con facilidad en sistemas informativos heterogéneos.

## 1.2. Estructuración y sintaxis

En un documento XML las etiquetas distinguen la estructura y el contenido informativo del documento. Los documentos en formatos basados en el etiquetado generalizado no contienen ninguna información que indique cómo se debe de presentar la información.

En la especificación que define W3C se usa el término *texto* para hacer referencia en forma conjunta a tantas etiquetas como al contenido textual.<sup>3</sup>

Para el contenido textual se utiliza el término *datos de carácter*. Para diferenciar dentro de un documento XML las etiquetas de los datos de carácter, la especificación establece la utilización de los caracteres reservados **<, >, & y ;**.

En el lenguaje XML se usa el término *elemento* para hacer referencia a cada uno de los componentes estructurales o secciones de un documento XML, un elemento queda delimitado por una etiqueta de inicio y una etiqueta de fin. **Por ejemplo:**

```
<unidades>texto para unidades </unidades>
```

En este ejemplo el elemento sería *unidades*. La etiqueta de inicio sería **<unidades>**, su contenido es **texto para unidades** y su etiqueta de cierre quedaría como **</unidades>**. Las etiquetas de cierre deben contener el carácter **/** antes del nombre del elemento.

Para referirnos al nivel de detalle con el que destacamos o diferenciamos los contenidos informativos del documento mediante la inclusión de etiquetas utilizamos el término *granularidad*. De esta manera, se puede distinguir un elemento y, dentro de éste, intercalar nuevas etiquetas para denotar con mayor precisión contenidos informativos y estructurales adicionales.

```
<persona>
  <nombre>Jorge Alberto Montalvo Olvera</nombre>
  <edad>32</edad>
  <fechanacimiento>18 julio 1981</fechanacimiento>
</persona>
```

*Documento XML sin granularidad*

<sup>3</sup> <http://www.w3.org/TR/2008/REC-xml-20081126/>

```

<persona>
  <nombrecompleto>
    <nombre>Jorge</nombre>
    <apellidopaterno>Montalvo</apellidopaterno>
    <apellidomaterno>Olvera</apellidomaterno>
  </nombrecompleto>
  <edad>32</edad>
  <fechanacimiento>
    <dia>18</dia>
    <mes>julio</mes>
    <anio>1981</anio>
  </fechanacimiento>
</persona>

```

*Documento XML con granularidad*

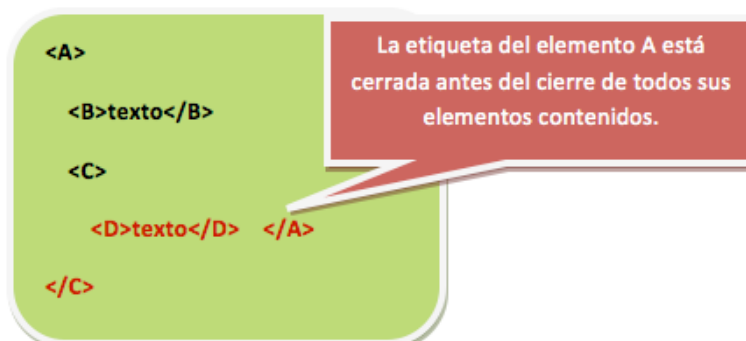
Dentro de un documento XML todos los elementos que lo conforman deben estar comprendidos en un único elemento: **elemento documento raíz**. Dicho elemento constituye la jerarquía de la estructura de los documentos XML, este tiene las siguientes características:

- Contiene todos los elementos del documento XML
- No está contenido en ningún otro elemento
- El nombre de este elemento debe coincidir con el nombre que se haya utilizado para designar al tipo de documento en cuestión.

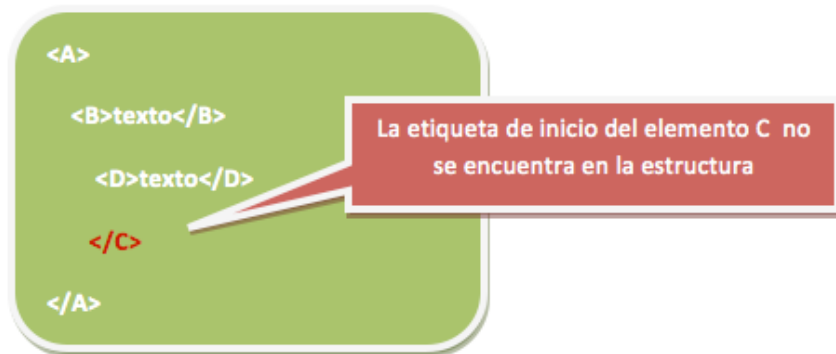
Para que las aplicaciones informáticas sea capaces de interpretar las etiquetas que identifican a cada elemento y así poder deducir la estructura jerárquica del documento XML para su posterior procesamiento, es necesario que las etiquetas estén anidadas correctamente.

Para que las etiquetas estén anidadas correctamente deben seguir las siguientes restricciones:

- La etiqueta de fin de un elemento A no puede escribirse hasta que no se haya introducido todas las etiquetas de cierre de los elementos cuya etiqueta de inicio esté situada después de la etiqueta de apertura del elemento A.



- No se puede introducir una etiqueta de cierre de un elemento si no se ha introducido antes su marca de inicio.



- Para todos los elementos no vacíos del documento se debe incluir una etiqueta de inicio y una de fin.
- Un documento XML debe incluir un prólogo que puede estar formado por dos secciones: la declaración XML y la declaración de tipo de documento. Tras el prólogo se encuentra la llamada **instancia del documento**. La instancia del documento es el contenido del documento propiamente dicho.

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

<!DOCTYPE note [
  <!ENTITY nbsp "&#xA0;">
  <!ENTITY writer "Writer: Donald Duck">
  <!ENTITY copyright "Copyright: W3S pols.">
]>

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
  <footer>&writer;&nbsp;&copyright;</footer>
</note>
```

Declaración XML

Declaración de tipo de Documento

- La declaración XML es obligatoria, y debe siempre aparecer al comienzo del documento. Ésta permite saber a la aplicación encargada de procesar el documento que se trata de un documento en formato XML. La declaración XML se escribe al comienzo del documento utilizando la siguiente sintaxis.

```
<?xml versión="1.0"?>
```

- La declaración de tipo de documento es opcional. El conjunto de elementos que se pueden incluir en un documento de un tipo determinado, el orden en que deben aparecer y cómo deben anidarse se encuentran en la llamada definición de tipo de documento o DTD.

- La declaración de tipo de documento indica a la aplicación que procesa el documento XML qué tipo de documento va a procesar.

**<!DOCTYPE nombre...>**

- La palabra nombre será sustituida por el nombre del tipo de documento en cuestión (nodo raíz del documento). Tras el nombre del nodo raíz, la declaración de tipo de documento puede contener la DTD (Document Type Definition) correspondiente a este tipo de documento, o en todo caso una referencia a un recurso externo que contenga la DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<diplomado>
  <nombre>Desarrollo de sistemas con tecnologia Java</nombre>
  <tecnologia>java</tecnologia>
  <modulos>
    <modulo>Programación orientada a objetos con JAVA</modulo>
    <modulo>Metodologías de análisis y desarrollo de sistemas
    orientadas a objetos</modulo>
    <modulo>Java para aplicaciones de escritorio</modulo>
    <modulo>Bases de datos con JDBC</modulo>
    <modulo>Uso de XML en la creación de interfaces web e intercambio
    de información</modulo>
    <modulo>Componentes Java para aplicaciones web</modulo>
    <modulo>Enterprise Java Beans</modulo>
  </modulos>
  <imparte>DGTIC UNAM</imparte>
</diplomado>
```

*Ejemplo de un documento XML bien formado*

### 1.3. Estructuración de contenidos XML

La especificación XML diferencia entre la estructura lógica y la estructura física de un documento. La estructura lógica está determinada por el conjunto de elementos que se usan en el documento y las relaciones jerárquicas que se establecen entre ellos. La estructura física está determinada por el número de archivos que constituye un documento XML. Un documento XML puede estar conformado por distintos archivos: recursos externos en formato binario, e incluso por múltiples archivos XML que se presentan en pantalla como si se tratase de un único archivo.

Los componentes de un documento XML son los siguientes:

Comentarios.- Los comentarios en los documentos XML empiezan por `<!--` y acaban por `-->`. Pueden contener cualquier cadena de texto excepto el literal `--`, además de que se pueden colocar en cualquier parte del documento.

```
<!-- esto es un comentario -->
```

Secciones CDATA.- Le indica al parser<sup>4</sup> que ignore todos los caracteres de etiquetas que se encuentren en el interior de éstas. Son muy útiles cuando se quiere visualizar código XML como parte del texto. Todos los caracteres que existan en esta sección son pasados directamente a la aplicación sin interpretación.

```
<![CDATA[      <!ENTITY amp "&"> <!-- &= ampersand -->
<CODIGO>
    *p=&q->campo;
    a=(x<y)?33:44;
</CODIGO>
]]>
```

Elementos.- Son las etiquetas más usadas en un documento XML. Están delimitadas por los símbolos `<` y `>`, si el contenido de la etiqueta es vacío `<` y `/>`. Las etiquetas de apertura pueden tener atributos, los cuales deben ser pares nombre/valor **nombre\_atributo="valor"**.

```
<nombre id="1" >Jorge Alberto</nombre>
```

Referencias a Entidades.- Las referencias en XML se utilizan como representación alternativa de los caracteres especiales, además pueden emplearse para incluir el contenido de otros documentos o para hacer referencia a un trozo de texto repetitivo. Su sintaxis es `&xxx`, donde `xxx` es el nombre de la entidad, y `&xxx` es la manera de referirse a la entidad.

```
&acute;      representa al símbolo é
```

Existe una referencia a entidades especiales, denominada referencia a caracteres. Ésta se usa para representar caracteres que no pueden ser escritos desde el teclado. No tiene un nombre de cadena sino que su nombre es, un número decimal o un número hexadecimal.

```
&#38;      símbolo ampersand
```

Se pueden crear constantes, para ello usamos la definición real de entidad.

```
<!ENTITY amp "&#38;";>
```

Para referenciarlo sería: `&amp;`

Atributos.- Para complementar a un elemento se hace uso de los atributos, los cuales dan información adicional para el procesamiento del documento. Las características de los atributos son:

- Tienen un nombre o un identificador

<sup>4</sup> Analizador sintáctico que indica si un archivo XML está bien formado.

- Almacenan un tipo de dato, o un valor de un conjunto de datos
- Pueden contener un carácter especial, obligatorio o fijo

Los atributos pueden recoger datos de los siguientes tipos en XML: CDATA, ENTITY, ID, IDREF.<sup>5</sup>

CDATA. Caracteres que no contienen etiquetas.

ENTITY. Nombre de la entidad

ID. Identificador único, se usa para identificar elementos

IDREF. Es el valor de un atributo ID de otro elemento, que debe existir en el documento XML.

---

<sup>5</sup> <https://sites.google.com/site/todoxmldtd/referencia/referencia-para-descripciones-xml/referencia-de-dtlds/05-declaracion-de-tipo-de-atributo-de-dtlds>



# 2 FUNDAMENTOS DE XML

Un documento XML bien formado, es el que cumple con las siguientes reglas:

- Tiene un documento raíz
- Todos los elementos tienen una etiqueta de inicio y de cierre
- Hay distinción entre mayúsculas y minúsculas
- Los elementos deben estar bien anidados
- Los atributos deben ir entre comillas

En el lenguaje XML se tienen dos opciones para definir las reglas de estructuración: DTD y XML Schema.

## 2.1. Creación de un archivo XML

Para la creación de un documento XML, se deben seguir las reglas anteriores, además de tener bien definida la información que se quiere estructurar. Una vez bien definida la estructura, se requiere de un editor de texto, aunque ya existen numerosos editores para XML, por lo que solo es cuestión de decidir cuál programa de edición utilizar. Por último se debe guardar el archivo creado con extensión .xml, para una mejor referencia de la información, se recomienda guardar el archivo con el nombre del elemento raíz de la estructura de información.

## 2.2. Creación de archivos DTD (Document Type Definition)

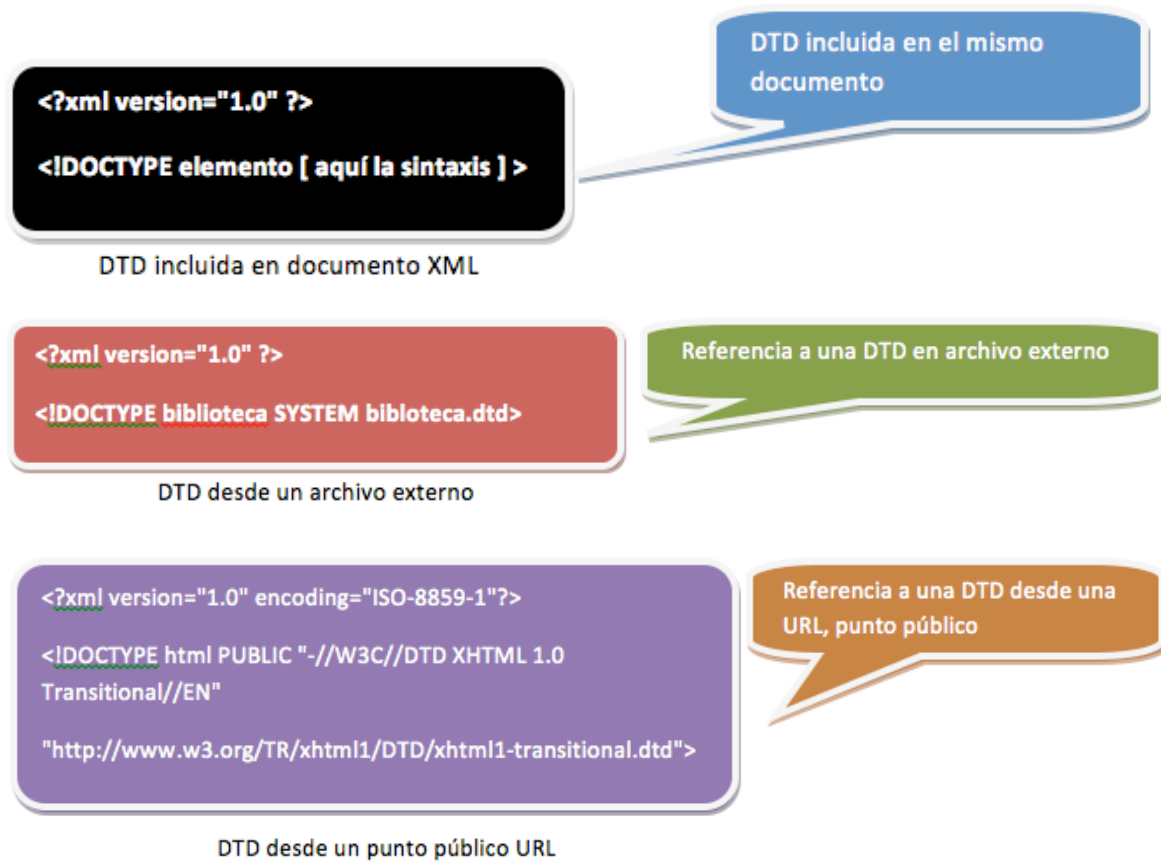
Las reglas específicas a cada tipo de documento se definen en una DTD (Document Type Definition). La DTD contiene información relativa a los elementos que se pueden utilizar en un tipo de documento específico: los elementos que son opcionales y los que no lo son, los elementos que se pueden repetir en más de una ocasión y los que aparecen una sola vez. Así como el orden en que deben escribir los elementos y la manera en que se deben anidar los elementos.

Dentro de la DTD también se define:

- Los atributos válidos para cada elemento
- El tipo de dato que puede tener cada atributo
- El carácter obligatorio u opcional de los atributos
- Las entidades que se pueden referenciar en el documento
- Los recursos externos de tipo XML, los que pueden ser procesados

- Los recursos externos no XML que no pueden ser procesables.

La DTD puede ser incluida en el mismo documento, definida en un archivo externo (archivo con extensión .dtd) y en acceso público (en una dirección url).



Una DTD está constituida por una o más declaraciones, con las cual se describe la estructura de un documento XML. Hay cuatro tipos de declaraciones: declaraciones de entidades, declaraciones de notaciones, declaraciones de elementos y declaraciones de atributos.

**Declaraciones de entidades.-** Las declaraciones de entidades internas siguen la siguiente sintaxis:<sup>6</sup>

```
<!ENTITY nombreEntidad "valorEntidad">
```

Cuando se trata de entidades externas se tienen dos casos: la entidad hace referencia a un archivo de texto externo y en este caso la entidad es sustituida por el contenido del archivo, estas pueden ser entidades de sistema o entidad pública.

```
<!ENTITY nombreEntidad SYSTEM "uri"> declaración de entidad de sistema
```

```
<!ENTITY nombreEntidad PUBLIC "fpi" "uri"> declaración de entidad publica
```

<sup>6</sup> [http://www.mclibre.org/consultar/xml/lecciones/xml\\_dtd.html](http://www.mclibre.org/consultar/xml/lecciones/xml_dtd.html)

Cuando la referencia a un archivo externo no es texto, podría ser una imagen, la entidad no se sustituye por el contenido del archivo. Para este caso la declaración de la entidad quedaría:

```
<!ENTITY nombreEntidad SYSTEM "uri" NDATA tipo>
```

En todos los casos anteriores:

- Uri (Identificador uniforme de recursos) la ruta al archivo el cual es accesible en una red o sistema. Aunque se acostumbra llamar URL a todas las direcciones web, URI es un identificador más completo y por eso es recomendado su uso en lugar de la expresión URL.
- tipo es el tipo de archivo
- fpi es un identificador público formal

**Declaración de notaciones.-** Las notaciones en el lenguaje XML se usan para definir entidades externas las cuales no se procesaran. Para poder hacer referencia a estas entidades se utiliza el nombre de la entidad directamente.

**Declaración de elementos.-** Para la declarar un elemento se sigue la siguiente sintaxis:

```
<!ELEMENT nombreElemento (contenido)>
```

Donde nombreElemento es el nombre del elemento y contenido es una expresión la cual indica el tipo de contenido de dicho elemento. El contenido de un elemento se define mediante EMPTY, #PCDATA o ANY.

**EMPTY.-** indica que el elemento es vacío, esto es, no puede tener contenido. Se puede escribir de la siguiente manera:

```
<!DOCTYPE ejemplo [
  <!ELEMENT elemento EMPTY>
]>

<elemento></elemento> Etiqueta de inicio y etiqueta de cierre sin nada entre
ellos, ni siquiera espacio
<elemento/> Etiqueta vacía
<elemento>Esto es un ejemplo</elemento> <!-- No es correcto porque contiene texto -->
<ejemplo><a></a></ejemplo> <!-- No es correcto porque contiene un elemento <a -->
```

**PCDATA.-** indica que el elemento puede contener datos de tipo carácter, números o fechas.

```
<!DOCTYPE elemento [
  <!ELEMENT elemento (#PCDATA)>
]>

<elemento />
<elemento>Esto es un ejemplo</elemento>
```

```
<elemento><a></a></elemento> <!-- No es correcto porque contiene un elemento <a> -->
```

**ANY.-** indica que no hay restricción sobre el contenido del elemento.

```
<!DOCTYPE elemento [
  <!ELEMENT elemento ANY>
]>
< elemento />
< elemento >Esto es un ejemplo</ elemento >
< elemento ><a></a></ elemento >
```

Un elemento puede contener otros elementos declarados en la DTD. En la definición de contenido de un documento se puede indicar que elementos se pueden incluir dentro del elemento que se declara. También se indica:

- Si estos elementos deben incluirse obligatoriamente o no (?)

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo (a, b?)>
  <!ELEMENT a EMPTY>
  <!ELEMENT b EMPTY>
]>
<ejemplo><a /></ejemplo>
<ejemplo><b /></ejemplo> <!-- No es correcto porque falta el elemento <a /> -->
```

- Si los elementos contenidos pueden aparecer más de una vez (\*)

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo (a*, b)>
  <!ELEMENT a EMPTY>
  <!ELEMENT b EMPTY>
]>
<ejemplo><b /></ejemplo>
<ejemplo><a /><b /></ejemplo>
<ejemplo><b /><a /></ejemplo> <!-- No es correcto porque el elemento <a /> aparece después de <b /> -->
```

- El elemento tiene que aparecer una o más veces (+)

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo (a+, b)>
  <!ELEMENT a EMPTY>
  <!ELEMENT b EMPTY>
]>
<ejemplo><a /><a /><b /></ejemplo>
```

```
<ejemplo><b /></ejemplo>  <!-- No es correcto porque falta el elemento <a /> -->
```

- Si no se indica ninguno de los caracteres anteriores tras el nombre del elemento, se indica que el elemento deberá aparecer una sola vez y no se puede repetir.
- Si los elementos aparecen mediante comas, deberán aparecer en el mismo orden que se hayan escrito.
- Si los elementos se separan mediante el carácter | (pipe), se podrá escribir uno de ellos.

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo (a | b)>
  <!ELEMENT a EMPTY>
  <!ELEMENT b EMPTY>
]>
<ejemplo><a /></ejemplo>
<ejemplo><b /></ejemplo>
<ejemplo><a /><b /></ejemplo>  <!-- No es correcto porque están los dos elementos -->
<ejemplo></ejemplo>  <!-- No es correcto porque no hay ningún elemento -->
```

En un modelo de contenido mixto, el elemento podrá contener datos de tipo PCDATA y otros elementos.

```
<!ELEMENT algo (#PCDATA | algomas)>
```

Los atributos se utilizan en XML para matizar el significado o alcance de los elementos. Todos los atributos están vinculados a un elemento, deben tener un tipo de dato asociado. Pueden tener un valor por defecto, pueden ser obligatorios o no y deberán ser declarados en la DTD.

```
<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorInicialAtributo >
```

Los tipos de datos de atributos son: CDATA, enumeraciones, ID, IDREF, IDEREFS , ENTITY, NMTOKEN y NMTOKENS.

**CDATA.-** cadenas de caracteres, fechas y números.

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
  <!ATTLIST ejemplo color CDATA #REQUIRED>
]>
<ejemplo color=""></ejemplo>
<ejemplo></ejemplo>  <!-- No es correcto porque falta el atributo "color" -->
<ejemplo color="azul marino #000080"></ejemplo>
```

Enumeraciones.- el atributo solo puede contener uno de los valores de la lista. La lista estará entre paréntesis con los valores separados por |.

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
```

```

    <!ATTLIST ejemplo color (azul|blanco|rojo) #REQUIRED>
  ]>

  <ejemplo color=""></ejemplo>
  <ejemplo color="azul"></ejemplo>
  <ejemplo color="verde"></ejemplo> <!-- No es correcto porque "verde" no está en la lista de valores -->

```

ID.- el valor del atributo deberá ser único y no se puede repetir en algún otro elemento o atributo definido en la DTD.

```

<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo (libro*)>
  <!ELEMENT libro (#PCDATA) >
  <!ATTLIST libro codigo ID #REQUIRED>
]>

<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>
</ejemplo>

<ejemplo>
  <libro codigo="1">Poema de Gilgamesh</libro> <!-- No es correcto porque el valor de un atributo de tipo ID no puede empezar con un número -->
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>
</ejemplo>

<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <libro codigo="L1">Los preceptos de Ptah-Hotep</libro> <!-- No es correcto porque el valor "L1" ya se ha utilizado -->
</ejemplo>

```

IDREF.- el valor del atributo debe coincidir con el valor del atributo ID de otro elemento.

```

<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo ((libro|prestamo)*)>
  <!ELEMENT libro (#PCDATA) >
  <!ATTLIST libro codigo ID #REQUIRED>
  <!ELEMENT prestamo (#PCDATA) >
  <!ATTLIST prestamo libro IDREF #REQUIRED>
]>

```

```

<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <prestamo libro="L1">Numa Nigerio</prestamo>
</ejemplo>
<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <prestamo libro="L2">Numa Nigerio</prestamo>  <!-- No es correcto porque el valor "L2" no
es ID de ningún elemento -->
</ejemplo>
IDREFS.- el valor del atributo es una serie de valores separados por espacios
que coinciden con el valor del atributo ID de otros elementos.
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo ((libro|prestamo)*)>
  <!ELEMENT libro (#PCDATA) >
  <!ATTLIST libro codigo ID #REQUIRED>
  <!ELEMENT prestamo (#PCDATA) >
  <!ATTLIST prestamo libro IDREFS #REQUIRED>
]>

<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>
  <prestamo libro="L1 L2">Numa Nigerio</prestamo>
</ejemplo>

<ejemplo>
  <libro codigo="L1">Poema de Gilgamesh</libro>
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>
  <prestamo libro="L3">Numa Nigerio</prestamo>  <!-- No es correcto porque el valor "L3" no
es ID de ningún elemento -->
</ejemplo>

```

ENTITY.- toma como valor el identificador de una entidad externa no procesable en la DTD

NMTOKEN.- toma como valor una secuencia de caracteres que no pueden incluir espacios en blanco. Puede contener de A-Z, a-z, 0-9, (-), (\_), (.) y (;)

```

<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
  <!ATTLIST ejemplo color NMTOKEN #REQUIRED>
]>

<ejemplo color="azul-marino"></ejemplo>

```

```
<ejemplo color="1"></ejemplo>
<ejemplo color="azul marino"></ejemplo> <!-- No es correcto porque hay un espacio en blanco -->
```

NMTOKENS.- toma como valor una secuencia de NMTOKEN separado entre sí por espacios en blanco.

Los atributos pueden ser opcionales u obligatorios. Con el fin de indicar en la declaración de una lista de atributos que un atributo es opcional u obligatorio, se utilizan respectivamente las palabras reservadas #IMPLIED y #REQUIRED

```
#REQUIRED
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
  <!ATTLIST ejemplo color CDATA #REQUIRED>
]>

<ejemplo></ejemplo> <!-- No es correcto porque falta el atributo "color" -->
<ejemplo color="amarillo"></ejemplo>

#IMPLIED
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
  <!ATTLIST ejemplo color CDATA #IMPLIED>
]>

<ejemplo color=""></ejemplo>
<ejemplo></ejemplo>
<ejemplo color="azul marino #000080"></ejemplo>
```

Un tipo especial de atributo es el de los atributos fijos. Éste es aquel que, siempre que aparece en el documento debe tomar el mismo valor. Para indicar en la DTD que un atributo es fijo, se utiliza la palabra reservada #FIXED

```
<!DOCTYPE ejemplo [
  <!ELEMENT ejemplo EMPTY>
  <!ATTLIST ejemplo color CDATA #FIXED "verde">
]>

<ejemplo></ejemplo>
<ejemplo color=""></ejemplo> <!-- No es correcto porque el atributo "color"
no tiene el valor "verde" -->
<ejemplo color="amarillo"></ejemplo> <!-- No es correcto porque el atributo
"color" no tiene el valor "verde" -->
<ejemplo color="verde"></ejemplo>
```



## 2.3. XML Schema

Un esquema XML es un documento XML que utiliza unos elementos y atributos específicos. Por lo tanto, un esquema XML puede tratarse como cualquier otro documento XML. Los esquemas XML equivalen a las DTD. Se puede decir que esquemas XML y DTD son formas alternativas para representar la estructura de un tipo de documento y las restricciones que sus instancias deben cumplir para ser válidos.

Los motivos por los que se propuso una alternativa a las DTD son:

- La complejidad de las DTD. Es necesario un mecanismo de representación más sencillo y evitar la necesidad de conocer dos sintaxis diferentes: XML y DTD
- La necesidad de indicar con mayor precisión el tipo de información que puede contener un elemento o un atributo, ofreciendo la posibilidad de utilizar tipos de datos más complejos y específicos que los que son utilizados en la DTD.
- Los esquemas XML utilizan la sintaxis del lenguaje XML en lugar de la sintaxis de la DTD. Así, un esquema que defina un tipo de documento específico será también un documento XML, y que debe cumplir con la condición de estar bien formado.

Los esquemas XML cumplen con la misma función que las DTD. Para que un documento XML sea válido, deberá cumplir todas las restricciones que se señalan en el esquema XML. En un esquema XML se indicará: los elementos que pueden aparecer en un documento, los elementos que son repetibles, opcionales u obligatorios y el orden de los elementos así también indicando cómo pueden anidarse o contenerse unos a otros. Además se indicarán las reglas para los atributos, definiendo qué atributos se pueden asociar a cada elemento y el tipo de datos que deben contener, qué atributos son opcionales u obligatorios y por último qué valores por defecto tendrán dichos atributos.

```
<?xml version="1.0" encoding="utf-8"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schema-microsoft-com:datatypes">
  <AttributeType name="id" dt:type="string" required="yes"/>
  <ElementType name="materia" dt:type="string" content="textOnly" />
  <ElementType name="lista-materias" content="eltOnly">
    <element type="materia" />
    <attribute type="id" />
  <ElementType/>
</Schema>
```

Los esquemas XML manejan el enfoque modular que utiliza POO. Los tipos de datos tienen en XML Schema la función de las clases en la POO. Se tiene la ventaja de construir tipos de datos de tipos predefinidos, agrupando elementos y atributos de una determinada forma y con mecanismos de extensión parecidos a la herencia. Los tipos de datos se clasifican en función de los elementos y atributos que contienen. Los tipos de datos que maneja XML Schema son: tipos de datos simples y tipos de datos complejos.

Datos simples.- estos no tienen elementos hijos ni atributos, como son: tipos predefinidos de XML, string, double, boolean, etc. Lista de datos separados por espacios y de tipo unión que son derivados de la unión de tipos predefinidos.

Datos complejos.- son los que contienen elementos hijos y/o atributos. Pueden tener nombre o ser anónimos, los que tienen nombre pueden ser reutilizados dentro del mismo XML Schema. Además de que pueden mezclarse elementos y texto.

Los esquemas XML manejan el concepto de namespaces. Un namespace permite definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento. Además ayuda a evitar confusiones al momento de reutilizar código.

# 3 CONSTRUYENDO DOCUMENTOS VÁLIDOS

La Validación de un XML es la comprobación de que un documento en lenguaje XML está bien formado y cumple con una estructura determinada. Un documento bien formado sigue las reglas básicas de XML establecidas para el diseño de documentos. Un documento válido además respeta las normas dictadas por su DTD (definición de tipo de documento) o esquema utilizado.

La validación de un documento XML consta de verificar:

- Formatos nulos o valores fuera de rango o incorrectos en los datos
- Que la información obligatoria se encuentre en el documento
- La interpretación de los datos en el documento sean de la misma manera tanto por quien lo emite y quien lo interpreta.
- La estructura de los elementos y atributos así como sus posibles valores
- El orden de los elementos
- Que elementos y/o atributos son requeridos u opcionales

## 3.1. Validación

La manera de validar un documento XML puede ser mediante una DTD o un XML Schema las cuales son métodos muy usados en la validación de documentos XML.

Ejemplo de una validación mediante DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library [
<!ELEMENT library (DVD+)>
<!ELEMENT DVD (title, format, genre)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ATTLIST DVD id CDATA #REQUIRED>
]>
<library>
```

```

<DVD id="1">
  <title>Breakfast at Tiffany's</title>
  <format>Movie</format>
  <genre>Classic</genre>
</DVD>
<DVD id="2">
  <title>Contact</title>
  <format>Movie</format>
  <genre>Science fiction</genre>
</DVD>
<DVD id="3">
  <title>Little Britain</title>
  <format>TV Series</format>
  <genre>Comedy</genre>
</DVD>
</library>

```

Una herramienta de acceso libre para realizar una validación es la que proporciona la W3C en <http://validator.w3.org/><sup>7</sup> donde se cuenta con 3 variantes de validación, validar una sitio web indicando su URI, enviando un archivo físico y añadiendo directamente el código.

**W3C** Markup Validation Service  
Check the markup (HTML, XHTML, ...) of Web documents

Validate by URI   Validate by File Upload   **Validate by Direct Input**

**Validate by direct input**

Enter the Markup to validate:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- This XML document describes a DVD library -->
<!DOCTYPE library [
  <ELEMENT library (DVD+)>
  <ELEMENT DVD (title, format, genre)>
  <ELEMENT title (#PCDATA)>
  <ELEMENT format (#PCDATA)>
  <ELEMENT genre (#PCDATA)>
  <ATTLIST DVD id CDATA #REQUIRED>
]>
<library>
  <DVD id="1">
    <title>Breakfast at Tiffany's</title>
    <format>Movie</format>
    <genre>Classic</genre>
  </DVD>
  <DVD id="2">
    <title>Contact</title>
    <format>Movie</format>
    <genre>Science fiction</genre>
  </DVD>
  <DVD id="3">
    <title>Little Britain</title>
    <format>TV Series</format>
    <genre>Comedy</genre>
  </DVD>
</library>

```

▼ More Options

☒ Validate Full Document

*Servicio de validación XML en W3C*

<sup>7</sup> <http://validator.w3.org/>

Al realizar la validación en el servicio proporcionado por W3C obtenemos lo siguiente:

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below this, there are links for "Jump To: Notes and Potential Issues" and "Congratulations - Icons". A green banner states "This document was successfully checked as XML!". Below the banner, a table shows the validation details:

<b>Result:</b>	Passed, 1 warning(s)	
<b>Source:</b>	<pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;!-- This XML document describes a DVD library --&gt; &lt;!DOCTYPE library [   &lt;!ELEMENT library (DVD+)&gt;   &lt;!ELEMENT DVD (title, format, genre)&gt;   &lt;!-- ELEMENT title (#PCDATA) --&gt;   &lt;!-- ELEMENT format (#PCDATA) --&gt;   &lt;!-- ELEMENT genre (#PCDATA) --&gt;   &lt;!-- ATTLIST DVD id CDATA #REQUIRED --&gt; ]&gt; &lt;library&gt;   &lt;DVD id="1"&gt;</pre>	
<b>Encoding:</b>	utf-8	(detect automatically)
<b>Doctype:</b>	XML	(detect automatically)
<b>Root Element:</b>	library	

*Resultado de una validación XML*

Ejemplo de validación mediante un xml schema:

Se tiene la siguiente estructura xml

```
<?xml version="1.0" encoding="UTF-8"?>
  <book isbn="0836217462">
    <title> Being a Dog Is a Full-Time Job </title>
    <author>Charles M. Schulz</author>
    <character>
      <name>Snoopy</name>
      <friend-of>Peppermint Patty</friend-of>
      <since>1950-10-04</since>
      <qualification> extroverted beagle </qualification>
    </character>
    <character>
      <name>Peppermint Patty</name>
      <since>1966-08-22</since>
      <qualification>bold, brash and tomboyish</qualification>
    </character>
  </book>
```

Ahora se evalúa el documento anterior mediante el siguiente xml schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="friend-of" type="xs:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xs:element name="since" type="xs:date"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Una herramienta online para poder realizar la validación anterior es <http://www.utilities-online.info/xsdvalidation/>

[Get simple example](#)

XML	XSD Schema
<pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;book isbn="0836217462"&gt;   &lt;title&gt;     Being a Dog Is a Full-Time Job   &lt;/title&gt;   &lt;author&gt;Charles M. Schulz&lt;/author&gt;   &lt;character&gt;     &lt;name&gt;Snoopy&lt;/name&gt;     &lt;friend-of&gt;Peppermint Patty&lt;/friend-of&gt;     &lt;since&gt;1950-10-04&lt;/since&gt;     &lt;qualification&gt;       extroverted beagle     &lt;/qualification&gt;   &lt;/character&gt;   &lt;character&gt;     &lt;name&gt;Peppermint Patty&lt;/name&gt;     &lt;since&gt;1966-08-22&lt;/since&gt;     &lt;qualification&gt;bold, brash and tomboyish&lt;/qualification&gt;   &lt;/character&gt; &lt;/book&gt;</pre>	<pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"&gt;   &lt;xs:element name="book"&gt;     &lt;xs:complexType&gt;       &lt;xs:sequence&gt;         &lt;xs:element name="title" type="xs:string"/&gt;         &lt;xs:element name="author" type="xs:string"/&gt;         &lt;xs:element name="character" minOccurs="0" maxOccurs="unbounded"&gt;           &lt;xs:complexType&gt;             &lt;xs:sequence&gt;               &lt;xs:element name="name" type="xs:string"/&gt;               &lt;xs:element name="friend-of" type="xs:string" minOccurs="0" maxOccurs="unbounded"/&gt;               &lt;xs:element name="since" type="xs:date"/&gt;               &lt;xs:element name="qualification" type="xs:string"/&gt;             &lt;/xs:sequence&gt;           &lt;/xs:complexType&gt;         &lt;/xs:element&gt;       &lt;/xs:sequence&gt;     &lt;/xs:complexType&gt;   &lt;/xs:element&gt;   &lt;xs:attribute name="isbn" type="xs:string"/&gt; &lt;/xs:schema&gt;</pre>
<input type="button" value="Check XML Well Formed"/>	<input type="button" value="Check XSD Validity"/>
<input type="button" value="Validate XML against XSD"/>	
Result	
The XML is Well Formed and Valid.	

Validación mediante XML Schema

# 4 DESPLIEGUE DE LA INFORMACIÓN

Inicialmente el lenguaje XML nos permite representar datos e información haciendo una clara separación entre la estructura de los mismos y su presentación. Para la parte de la presentación de los datos se hace uso de mecanismos externos como CSS, XLS, XLST y XPATH.

## 4.1. CSS

Las siglas CSS corresponden a Cascading Style Sheets la cual es el estándar para normalizar la creación de hojas de estilo para documentos HTML y XML. CSS define distintos niveles; CSS-1, CSS-2 y CSS-3. Cada nivel describe un conjunto de propiedades y características que deben de implementar las aplicaciones para soportar los distintos niveles de la especificación.

```

5  input.post, textarea.post
6  {
7      color: #000;
8      background: #F5F9FA;
9      border: #E6E6E6 solid 1px;
10     clear: both;
11     font-family: "Trebuchet MS", Trebuchet, Arial, Helve
12     font-size: 12px;
13 }
14 input:focus.post, textarea:focus.post
15 {
16     background: #FFFFFF;
17 }
18 #aviso_noph

```

*Ejemplo de código CSS<sup>8</sup>*

Las principales ventajas del uso de CSS son:

- Representaciones económicas de los documentos
- Representación homogénea de los documentos
- La posibilidad de generar múltiples presentaciones para un mismo contenido

<sup>8</sup> [http://l4c.me/uploads/sublime-text-2-css-code-1339689997\\_full550.png](http://l4c.me/uploads/sublime-text-2-css-code-1339689997_full550.png)



Una hoja de estilo CSS está formada por una o más reglas. Cada regla contiene las instrucciones de formateo que se deben aplicar a las instancias de un elemento determinado.

Cada instrucción se llama declaración, y en ella se asigna un valor a una propiedad. Una declaración tiene dos partes: el nombre de la propiedad y el valor que se le asigna.

El nombre de la propiedad se separa de su valor mediante dos puntos (:), si una regla contiene más de una declaración, éstas se separan por un punto y coma (;). Las declaraciones de una misma regla se encierran entre llaves { y }. Además las declaraciones van precedidas por un selector que indica a qué elementos del documento se tienen que aplicar las declaraciones.



*Declaración de una regla CSS<sup>9</sup>*

Los principales selectores que define CSS son:

- Universal. **\* {declaraciones}**
- Selector de tipo. **elemento {declaraciones}**
- Selector de grupo. **elemento1, elemento2 {declaraciones}**
- Elementos hijo. **elemento1 > elemento2 {declaraciones}**
- Selector basado en el atributo class. **elemento1.val{declaraciones}**
- Selector basado en el atributo ID. **elemento1#val{declaraciones}**

Documento XML pseudoSelector.xml haciendo la llamada a una hoja de estilo

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="ligas.css"?>
<documento xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">
  <titulo>Ejemplo con enlace hipertexto</titulo>
  <par>Esta es un documento de prueba</par>
  <par>
    Haz click en este <HTML:A HREF="http://www.java.sun.com">
```

<sup>9</sup> <http://3.bp.blogspot.com/-MzALHYfxSCA/UaoZ4UVqvLI/AAAAAAAAAKhI/ZWbhiuUMr-E/s1600/reglas+css.jpg>

```

                                enlace para visitar un buen sitio</HTML:A>
        </par>
</documento>

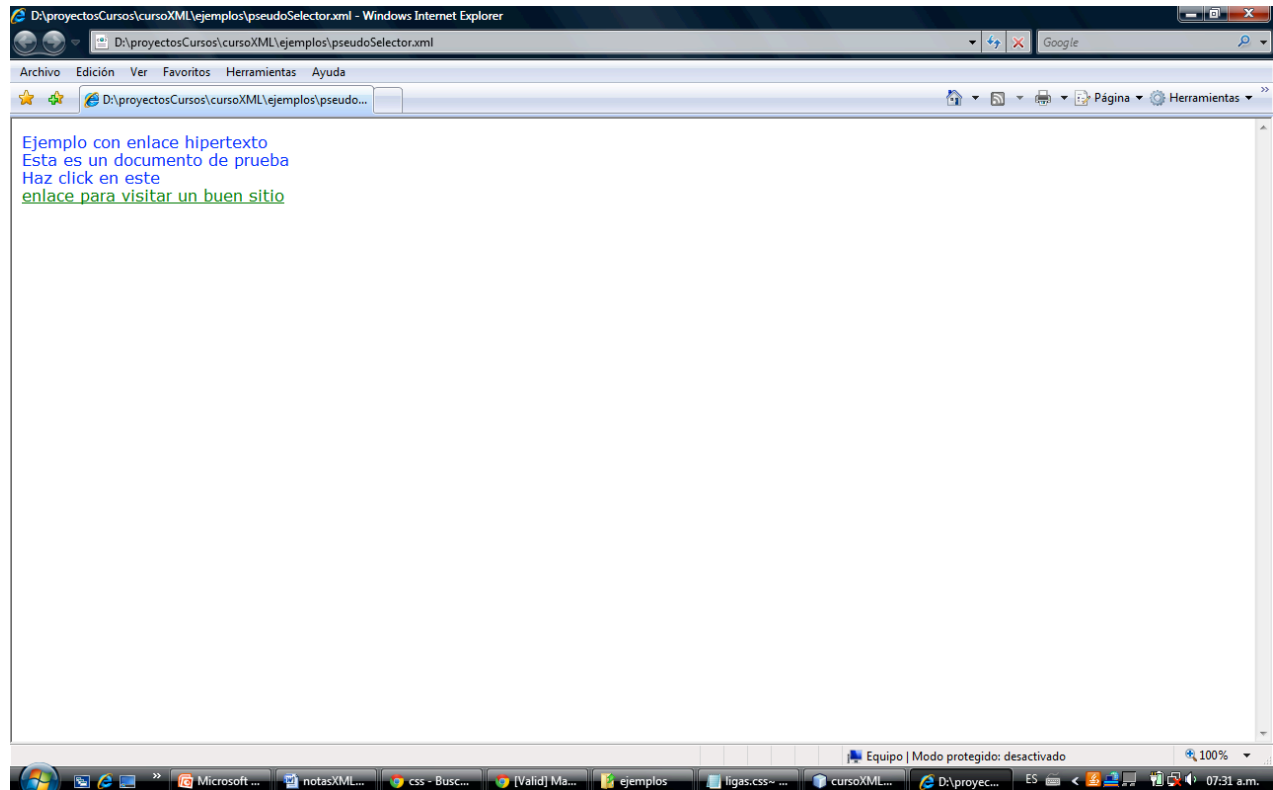
```

### Hoja de estilo ligas.css

```

* {
    display : block;
    font-family : Verdana;
    color : blue;
    font-size : 12pt
}
HTML\:A { color : green }
HTML\:A:active { color : red }
HTML\:A:visited { color : black }

```



*Resultado de la aplicación de un CSS*

## 4.2. XSL

XSL (siglas de Extensible Stylesheet Language, lenguaje extensible de hojas de estilo) es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio. Esta familia está formada por tres lenguajes:

- XSLT (Extensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación), que permite convertir documentos XML de una sintaxis a otra.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" version="4.0"/>
  <xsl:template match="/">
    <html><head><title>Vuelos de hoy</title></head>
      <body><h1>Horario de vuelos</h1>
        <table border="1">
          <tr><th>Numero Vuelo</th>
          <th>Origen</th>
          <th>Destino</th>
          <th>Hora</th></tr>
          <xsl:for-each select="//vuelo">
            <xsl:sort select="@origen"/>
            <tr><td><xsl:value-of select="@numero"/></td>
            <td><xsl:value-of select="@origen"/></td>
            <td><xsl:value-of select="@destino"/></td>
            <td><xsl:value-of select="@hora"/></td>
            </tr>
          </xsl:for-each>
        </table></body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

- XSL-FO (lenguaje de hojas extensibles de formateo de objetos), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF. Ejemplo de código XSL-FO

```

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:table table-layout="fixed" width="17.5cm">
    <xsl:for-each select="//tutorial">
      <fo:table-column column-width="3cm"/>
      <fo:table-column column-width="14.5cm"/>
      <fo:table-body>
        <fo:table-row height="3cm"> <!-- -->
          <fo:table-cell>
            <fo:block margin-left="1.5cm" margin-right="1.5cm">
              <xsl:value-of select="autor"/>
            </fo:block>
          </fo:table-cell>
        </fo:table-row>
        <fo:table-row height="3cm"> <!-- -->
          .....
        </fo:table-row>
      </fo:table-body>
    </xsl:for-each>
  </fo:table>
</fo:root>

```

- XPath, o XML Path Language, (no basada en XML) sirve para acceder o referirse a porciones de un documento XML. Ejemplo de una expresión XPath:
  - **/libreria/libro[precio>35.00]/titulo** Selecciona el título de todos los elementos libro del elemento librería que tengan precio mayor a 35.00

### 4.3. XSLT

Una hoja de estilo XSLT es un documento XML. Por lo tanto, debe tratarse de un documento XML bien formado y debe tener las siguientes reglas XML:

- Una hoja de estilo XSLT debe comenzar con una declaración XML.
- Después de la declaración XML, aparecerá el elemento raíz de la hoja de estilo: stylesheet.
- El elemento raíz contendrá a todos los demás elementos y debe ir precedido por el alias xsl correspondiente al espacio de nombres de las hojas de estilo XSLT.
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Para utilizar hojas de estilo XSLT en Internet Explorer, en vez del URI <http://www.w3.org/1999/XSL/Transform> del espacio de nombres XSLT, se debe escribir el URI: <http://www.w3.org/TR/WD-xsl>.

Tras las marcas desde inicio y fin del elemento raíz xsl:stylesheet, se pueden escribir las reglas de transformación propiamente dichas. El principal elemento que se utiliza en hojas de estilo XSLT es el elemento xsl:template. Cada elemento template define una regla de transformación que se aplicará sobre las instancias de los elementos del documento XML.

Para asociar una hoja de estilo XSLT a un documento XML, debemos añadir a éste una instrucción de procesamiento.

```
<?xml-stylesheet type="text/xsl" href="estilo.xsl"?>
```

### Ejemplo de un document XLST

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" version="4.0"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Propiedades</title>
      </head>
      <body>
        <h1>Propiedades Disponibles</h1>
        <xsl:for-each select="//propiedad">
          <xsl:if test="@id='1'">
            <h1><xsl:value-of select="direccion" /></h1>
            <h1>Es la mejor</h1>
          </xsl:if>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

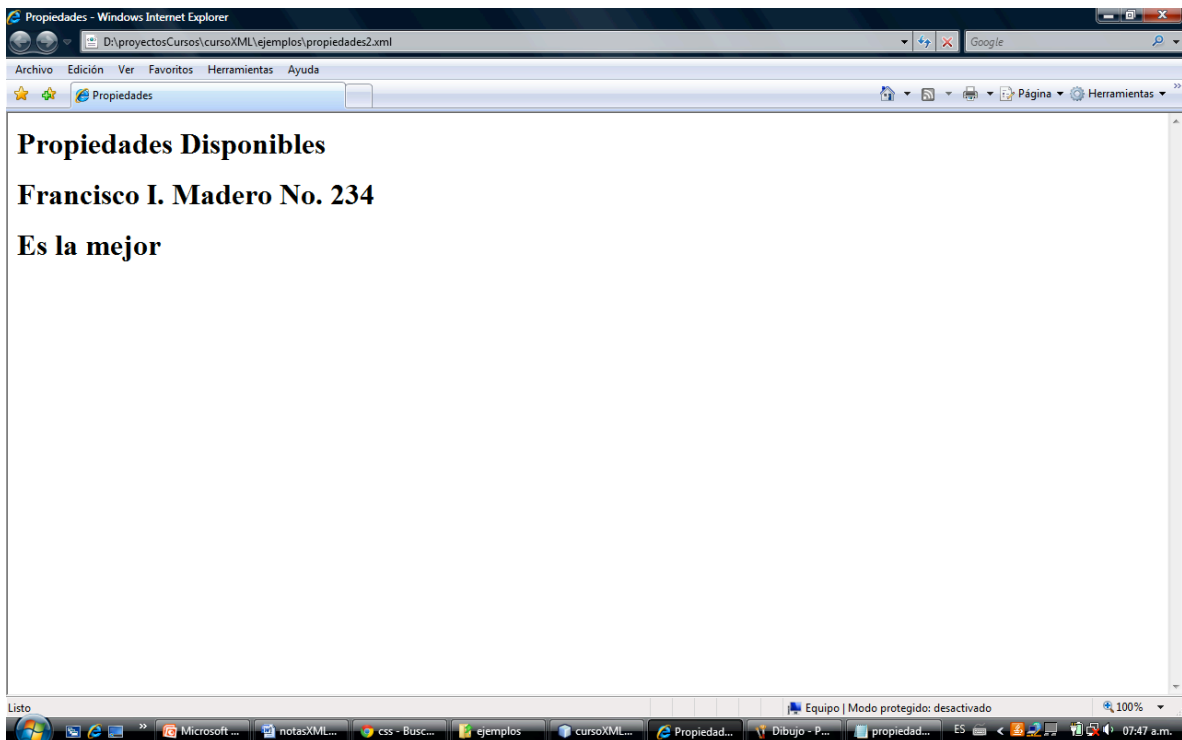
Aplicando el estilo anterior quedaría:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="propiedadesxsl.xsl"?>
<propiedades>
  <propiedad id="1">
    <direccion>Francisco I. Madero No. 234</direccion>
    <precio>$100,000</precio>
    <comentarios>Cerca del metro de linea 5</comentarios>
  </propiedad>
  <propiedad id="2">
    <direccion>Marte No. 15</direccion>
    <precio>$230,000</precio>
    <comentarios>Colonia muy tranquila</comentarios>
  </propiedad>
  <propiedad id="3">
    <direccion>Av. Tlahuac No. 4174</direccion>
```

```

<precio>$150,000</precio>
<comentarios>Buena vista de paisaje</comentarios>
</propiedad>
<propiedad id="4">
<direccion>Misterios No. 21</direccion>
<precio>$80,000</precio>
<comentarios>Barrio conflictivo</comentarios>
</propiedad>
<propiedad id="5">
<direccion>Pueblo Quieto No. 1000</direccion>
<precio>$600,000</precio>
<comentarios>Muy tranquilo</comentarios>
</propiedad>
<propiedad id="6">
<direccion>Sin nombre</direccion>
<precio>$300,000</precio>
<comentarios>Sin comentarios</comentarios>
</propiedad>
</propiedades>

```



*Resultado de la aplicación de un XSLT*

XSLT define los siguientes elementos para la transformación de los datos:

- **xsl:Stylesheet** Elemento principal, éste contiene todos los demás elementos para la transformación de los datos, define que el documento es una hoja de estilo XSLT (con el número de versión y el atributo namespace de XSLT).
- **xsl:template** Elemento que es usado para crear plantillas. El atributo **match="/"** asocia a la plantilla con la raíz del documento XML.
- **xsl:apply-templates** Es usada para aplicar una plantilla (template) sobre el elemento actual o sobre alguno de sus nodos hijos.
- **xsl:call-template** Invoca una plantilla por nombre.
- **xsl:transform** Elemento raíz que declara un documento XSL.
- **xsl:value-of** Puede ser usada para extraer los valores de un elemento XML y añadirlo a la salida de una transformación.
- **xsl:for-each** Puede ser usada para seleccionar cada elemento XML de un nodo específico.
- **xsl:if** Para introducir una orden condicional en el tratamiento del documento XML.
- **xsl:choose, xsl:when, xsl:otherwise** Para expresar múltiples condiciones contra el contenido de un documento XML.
- **xsl:sort** Ordena la salida y se usa dentro de la etiqueta xsl:for-each.
- **xsl:import, xsl:include** Importa (incluye) un archivo XSL al documento en instancia.
- **xsl:output** Imprime el valor de un variable creada.
- **xsl:variable** Define una variable, puede ser usada en un loop for-each.
- **xsl:param** Declara un parámetro con nombre que se puede utilizar dentro de un elemento `<xsl:stylesheet>` o un elemento `<xsl:template>`.
- **xsl:copy-of** Inserta ramas del árbol de nodos a una rama del árbol.
- **xsl:element** Crea un elemento de salida con el nombre especificado.
- **xsl:attribute** Accesa a un atributo de un elemento determinado.
- **xsl:text** Genera un nodo de texto a partir de una hoja de estilos.

## 4.4. XPATH

XPath es una especificación de la familia XML que indica la sintaxis que se debe utilizar para recuperar nodos y elementos de un documento XML. XPath permitirá seleccionar los elementos a los que se va a aplicar una hoja de estilo. Cuando se define una expresión XPath hay que tomar en cuenta:

- El contexto desde el cual se va a evaluar la expresión.
- La dirección de la expresión.
- Las condiciones que deben cumplir los nodos para ser recuperados.

XPath es un elemento importante en el estándar XSLT, en XPath no es posible crear documentos XSLT.

Una expresión XPath se evalúa tomando como base un nodo que sirve de contexto. El nodo que sirve de contexto es el nodo que está siendo procesado en el momento en el que se evalúa la expresión XPath. En XPath, hay siete tipos de nodos: elementos, atributos, texto, espacio de nombres, el procesamiento de instrucciones, comentarios, y los nodos del documento.

En XPath hay una relación entre los diferentes tipos de nodos, inicialmente se tiene el nodo padre o nodo raíz, esto es, cada elemento o atributo tiene un padre.

Nodos hijos.- Los nodos de elementos pueden tener cero, uno o más hijos.

Nodos hermanos.- Los nodos que tienen el mismo padre.

Nodos ancestros.- El padre de un nodo, el padre del padre, etc.

Nodos descendientes.- Los hijos de un nodo, hijos de los hijos, etc.

A partir del contexto en el que se evalúa una expresión, es posible indicar la dirección de la búsqueda.

Finalmente, además del contexto y la dirección, una expresión XPath es necesario indicar la condición que deben cumplir los nodos para ser recuperados. Esta condición puede ser:

- Que el nodo tenga un nombre determinado
- Que el nodo tenga un atributo determinado
- Que el nodo tenga un atributo que recoja un valor determinado
- Que el nodo tenga un elemento hijo determinado

XPath utiliza expresiones de ruta para seleccionar nodos en un documento XML. El nodo se selecciona siguiendo una ruta o pasos.<sup>10</sup>

Expresión	Descripción
Nombre del nodo	Selecciona todos los nodos con este nombre
/	Selecciona desde el nodo raíz
//	Selecciona nodos del documento desde el nodo actual que coincidan con la selección sin importar donde se encuentren
.	Selecciona el nodo actual
..	Selecciona el nodo padre del nodo actual
@	Selecciona los atributos

*Expresiones XPath*

<sup>10</sup> [http://www.w3schools.com/xpath/xpath\\_syntax.asp](http://www.w3schools.com/xpath/xpath_syntax.asp)



XPath hace uso de funciones llamadas predicados las cuales se utilizan para encontrar un nodo específico o un nodo que contenga un valor específico.

Expresión	Resultado
/biblioteca/libro[1]	Selecciona el primer elemento de libro que es el hijo del elemento biblioteca.  Nota: En IE 5,6,7,8,9 primer nodo es [0], pero de acuerdo con W3C, es [1].
/biblioteca/libro[last()]	Selecciona el último elemento de libro que es el hijo del elemento biblioteca.
/biblioteca/libro[last()-1]	Selecciona el penúltimo libro de elemento que es el hijo del elemento biblioteca.
/biblioteca/libro[position()<4]	Selecciona los tres primeros elementos del libro que son hijos del elemento biblioteca.
//titulo[@idioma='ingles']	Selecciona todos los elementos del título que tienen un atributo llamado idioma.
/biblioteca/libro[precio>35]	Selecciona todos los elementos de libros de la biblioteca que tienen un elemento de precio con un valor superior a 35.00.
/biblioteca/libro[precio>35]/titulo	Selecciona todos los elementos del título de los elementos de libros de la biblioteca de elementos que tienen un elemento de precio con un valor superior a 35,00.

*Expresiones XPath usando predicados*

Para realizar la selección de nodos desconocidos se debe realizar de la siguiente manera:

Expresion	Descripción
*	Coincide con cualquier nodo de elemento.
@*	Coincide con cualquier nodo de atributo.
Node()	Coincide con cualquier nodo de cualquier tipo

*Expresiones para cualquier coincidencia*

# 5 MODELOS DOM

**D**OM (Document Object Model) es una interfaz de programación (API) para documentos XML y HTML (XHTML). Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento. La interfaz de programación DOM puede ser utilizada desde varios lenguajes de programación, como Java y JavaScript.

En la actualidad existen tres especificaciones:

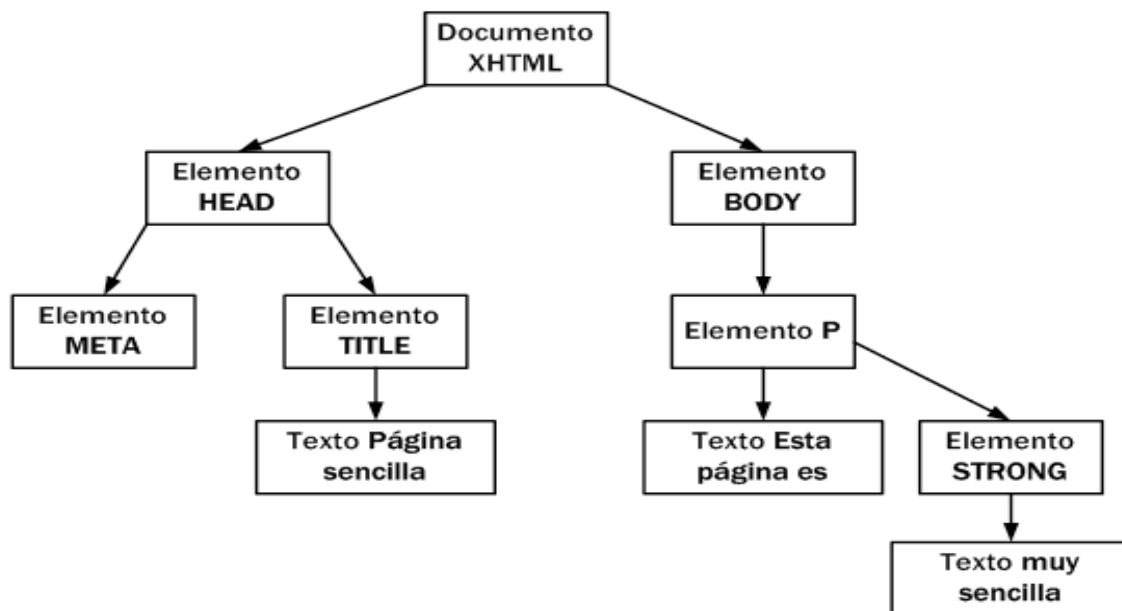
- DOM nivel 1 Consiste en dos módulos: DOM Core y DOM HTML. Este nivel es soportado completamente por Internet Explorer y FireFox
- DOM nivel 2 Está construido sobre el nivel 1 y consiste de 14 módulos. Entre los módulos más importantes se encuentran el módulo de manejo de eventos de usuarios y el módulo de hojas de estilo. Este último representa los estilos que se le pueden asignar a un documento. Este nivel es soportado parcialmente por Internet Explorer y FireFox.

La API de DOM es independiente de cualquier lenguaje de programación. De hecho, DOM se ha portado a la mayoría de lenguajes de programación comúnmente empleados.

Si se emplea la siguiente página HTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Página sencilla</title>
  </head>
  <body>
    <p>Esta página es <b>muy sencilla</b></p>
  </body>
</html>
```

La representación DOM de la página anterior sería la siguiente:



La página HTML se ha transformado en una jerarquía de nodos, en el que el nodo raíz es uno de tipo documento HTML. A partir de este nodo, existen 2 más en el mismo nivel formados por las etiquetas `<head>` y `<body>`. De cada uno de los anteriores sale otro nodo (`<title>` y `<p>` respectivamente). Por último, de cada nodo anterior sale otro de tipo "texto".

## 5.1. Nodos

Antes de poder utilizar la API de DOM, se construye de forma automática el árbol para poder ejecutar de forma eficiente todas esas funciones. De este modo, para utilizar DOM es imprescindible que la página web se haya cargado por completo, ya que de otro modo no existe el árbol asociado y las funciones DOM no pueden funcionar correctamente.

Los documentos XML (y HTML) tratados por DOM se convierten en una jerarquía de nodos. Los nodos que representan los documentos pueden ser de diferentes tipos. A continuación se mencionan los tipos más importantes:

- **Document** Nodo raíz de un documento HTML, es dueño de los demás nodos contenidos en el árbol jerárquico.
  - `documentObject.nodeName`
- **DocumentType** Provee de una interface para las entidades definidas en un documento XML

```
xmlDoc=loadXMLDoc("note_internal_dtd.xml");
x=xmlDoc.doctype.entities;
for (i=0;i<x.length;i++) {
    document.write("Nodename: " + x.item(i).nodeName);
    document.write("");
    document.write("Nodetype: " + x.item(i).nodeType);
    document.write("");
}
```

- Element Representa un elemento en un documento XML.
  - `elementNode.nodeName`
- Text Contenido textual de un elemento o un atributo.

```
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
document.write(x.data);
```

Los atributos son propiedades del elemento, no elementos secundarios del elemento. Es importante hacer esta distinción, debido a los métodos utilizados para desplazarse por los nodos relacionados, principales y secundarios del DOM.

## 5.2. Elementos DOM

Todo documento contiene uno o más elementos, cuyos límites están o bien delimitados por etiquetas iniciales y etiquetas finales, o bien, en el caso de documentos vacíos, por una etiqueta de elemento vacío. Cada elemento tiene un tipo, identificado por un nombre, y puede tener un conjunto de atributos. Cada atributo tiene un nombre y un valor.



```
<persona edad="32">
    <nombre>Carlos Fuentes</nombre>
</persona>
```

## 5.3. Trabajando con Nodos

El objeto Document brinda métodos para acceder a los elementos del documento HTML o XML, algunos de los cuales son:

- Element `getElementById (id_name)` : retorna el nodo cuyo atributo id es `id_name`

En el ejemplo siguiente, el elemento con identificador texto será obtenido y posteriormente se obtendrá su valor inicial.

```
<script>
    function getValue() {
        var x=document.getElementById("texto");
        alert(x.value);
    }
</script>
<input type='text' id='texto' value="mi valor" onclick="getValue()" />
```

- NodeList `getElementsByName (elemento_name)` : retorna la colección de nodos cuyo atributo nombre (name) es igual a `elemento_name`. En el siguiente ejemplo se imprime el valor de todos los elementos con el nombre de x.

```
var x=document.getElementsByTagName("x");
alert(x.length);
```

- `NodeList getElementsByTagName (tag_name)` : retorna la colección de nodos cuyo atributo etiqueta (tag) es igual a tag\_name. En el siguiente ejemplo se obtiene el primer elemento UL del documento.

```
var list=document.getElementsByTagName("UL")[0]
```

Los métodos DOM disponibles para la creación de nuevos nodos son los siguientes:

- `createAttribute(nombre)` Crea un nodo de tipo atributo con el nombre indicado.

```
var att=document.createAttribute("class");
att.value="HOLA MUNDO";
document.getElementsByTagName("H1")[0].setAttribute(att);
```

Antes de crear el atributo class

Después de crear el atributo class

HOLA MUNDO

HOLA MUNDO

- `createComment(texto)` Crea un nodo de tipo comentario que contiene el valor indicado

```
var c=document.createComment("Ejemplo de comentario");
document.body.appendChild(c);
```

El resultado es:

```
<!--Ejemplo de comentario-->
```

- `createElement(nombre_etiqueta)` Crea un elemento del tipo indicado en el parámetro nombre\_etiqueta.
- `createTextNode(texto)` Crea un nodo de tipo texto con el valor indicado como parámetro

```
var btn=document.createElement("BUTTON");
var t=document.createTextNode("CLICK ME");
btn.appendChild(t);
```

El resultado será:

CLICK ME

# 6 JAVA Y XML

Java contiene varios métodos para acceder a XML. La siguiente es una breve descripción de los métodos disponibles. Java proporciona la API DOM (Document Object Model). En DOM se accede al documento XML a través de un árbol de objetos. DOM se puede utilizar para leer y escribir archivos XML.

## 6.1. JDOM

JDOM es un modelo de objetos de documentos basado en Java de código abierto para XML que fue diseñado específicamente para la plataforma Java para que pueda tomar ventaja de sus características lingüísticas. JDOM se integra con Document Object Model (DOM) y API simple para XML (SAX), compatible con XPath y XSLT. Se utiliza analizadores externos para construir documentos.

La forma de trabajar de JDOM es muy similar al DOM, JDOM genera un árbol de nodos al momento de parsear un documento XML. Los tipos de nodos son similares a DOM. Al ser similar a DOM, JDOM no está modelado ni creado sobre DOM, solo es un modelo alternativo.

Ejemplo de aplicación de JDOM:

Como primer paso, se parte de un documento XML, en este caso, es una estructura xml de datos de una liga de futbol.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document      : ligaFutbol.xml
  Author       : jorge
  Description:
    Purpose of the document follows.
-->

<liga tipo="Champions League">
  <equipo>
    <club valoracion="10" ciudad="Bilbao">Athletic Club Bilbao</club>
    <presidente>Uria</presidente>
    <plantilla>
      <nombre>Julen Guerrero</nombre>
      <nombre>Joseba Etxeberria</nombre>
      <nombre>Ismael Urzaiz</nombre>
    </plantilla>
  </equipo>
  <equipo>
    <club valoracion="5" ciudad="Madrid">Real Madrid</club>
    <presidente>Mandamas</presidente>
    <plantilla>
      <nombre>Bota de oro</nombre>
      <nombre>Milloneti</nombre>
      <nombre>Canterano quemado</nombre>
    </plantilla>
    <conextranjeros/>
  </equipo>
  <arbitros>
    <nombre>No doy una</nombre>
    <nombre>Juan Perez</nombre>
  </arbitros>
</liga>
```

#### *Ejemplo de Archivo XML*

Como segundo paso, se debe parsear el documento (leer y verificar que esté bien formado el documento XML). Con JDOM para parsear un documento XML hace uso de SAX el cual procesa los documentos XML por bloques lo que permite no tener todo el documento en memoria.

Para parsear realizamos:

```

SAXBuilder builder = null;
Document doc = null;
try {
    builder = new SAXBuilder(false);
    doc=builder.build("ligaFutbol.xml");
} catch (JDOMException jdex) {
    System.out.println("Error al cargar el archivo xml"
        + jdex.getMessage());
}
}

```

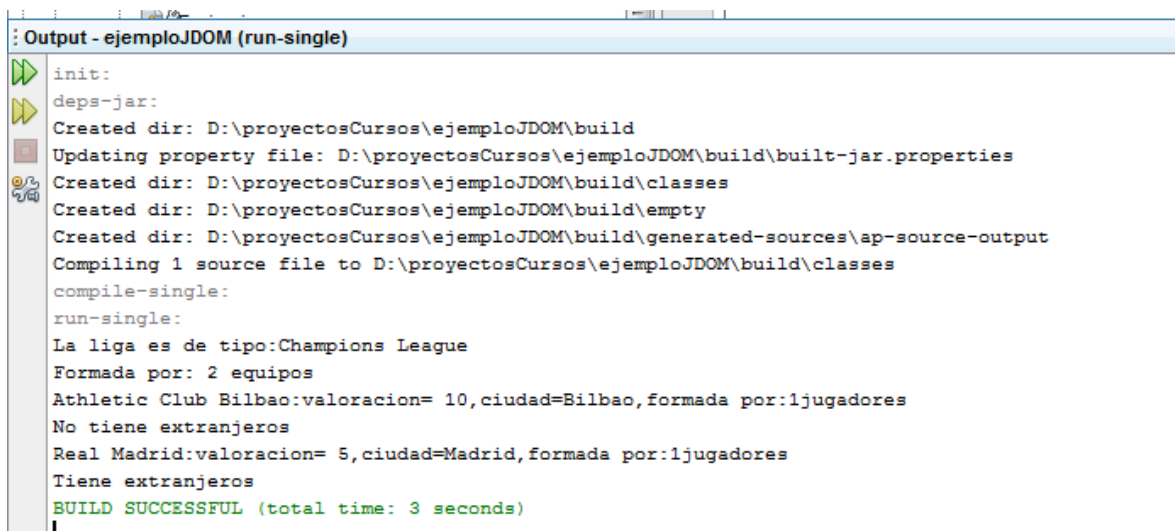
Una vez obtenido el documento raíz se recorre el árbol de nodos.

```

//todos los hijos que tengan como nombre plantilla
List equipos = raiz.getChildren("equipo");
System.out.println("Formada por: " + equipos.size() + " equipos");
Iterator i = equipos.iterator();
while (i.hasNext()){
    Element e = (Element) i.next();
    //primer hijo que tenga como nombre club
    Element club = e.getChild("club");
    List plantilla = e.getChildren("plantilla");
    System.out.println(club.getText() + ":" + "valoracion= " +
        club.getAttributeValue("valoracion") + "," +
        "ciudad=" + club.getAttributeValue("ciudad") + "," +
        "formada por:" + plantilla.size() + "jugadores");
    if (e.getChildren("conextranjeros").size() == 0)
        System.out.println("No tiene extranjeros");
    else System.out.println("Tiene extranjeros");
}

```

Resultado de la ejecución del código anterior:



```

Output - ejemploJDOM (run-single)
init:
deps-jar:
Created dir: D:\proyectosCursos\ejemploJDOM\build
Updating property file: D:\proyectosCursos\ejemploJDOM\build\build-jar.properties
Created dir: D:\proyectosCursos\ejemploJDOM\build\classes
Created dir: D:\proyectosCursos\ejemploJDOM\build\empty
Created dir: D:\proyectosCursos\ejemploJDOM\build\generated-sources\ap-source-output
Compiling 1 source file to D:\proyectosCursos\ejemploJDOM\build\classes
compile-single:
run-single:
La liga es de tipo: Champions League
Formada por: 2 equipos
Athletic Club Bilbao: valoracion= 10, ciudad=Bilbao, formada por: 1jugadores
No tiene extranjeros
Real Madrid: valoracion= 5, ciudad=Madrid, formada por: 1jugadores
Tiene extranjeros
BUILD SUCCESSFUL (total time: 3 seconds)

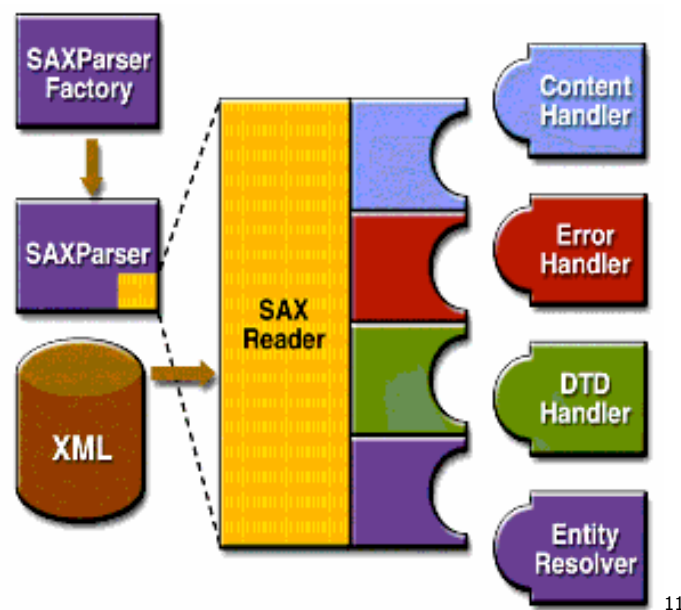
```



## 6.2. JAXP

El API Java JAXP permite analizar, transformar y validar documentos XML utilizando una API que es independiente de una implementación particular XML. JAXP proporciona una capa de conectividad que permitir a los desarrolladores proporcionar sus propias implementaciones sin introducir dependencias de código de la aplicación.

JAXP aprovecha la normas del analizador simple para XML Parsing (SAX) y Document Object Model (DOM), también soporta el estándar Extensible Stylesheet Language Transformations (XSLT), el cual permite controlar la presentación de los datos así como convertir los datos a otros documentos XML o en otros formatos como HTML. JAXP también proporciona soporte de espacio de nombres, que permite trabajar con DTDs que de otro modo podrían tener conflictos de nombres.



*Esquema de trabajo SAX para XML*

<sup>11</sup> <http://docs.oracle.com/javase/tutorial/figures/jaxp/intro/jaxpintro-saxApi.gif>

```

try {
    /*
    puede parecer un lío mezclar "SAXParser" y "XMLReader", pero solo es
    un problema de nomenclatura entre versiones. Se usa un SAXParser,
    que implementa el interface XMLReader (todos los parsers de XML lo
    hacen), porque en la version 1 de SAX ya se uso el termino "parser",
    y se mantiene por convencion.
    */
    XMLReader parser = new SAXParser();
    //añadimos al parser nuestro "ContentHandler", pasandole el vector de instancias.
    XMLHandler xmlHandler = new XMLHandler();
    xmlHandler.setInstancias(new ArrayList<Pagina>());
    parser.setContentHandler(xmlHandler);
    //y le decimos que empiece a procesar nuestro fichero de favoritos
    parser.parse("paginas.xml");
    for (Pagina pagina : xmlHandler.getInstancias()) {
        System.out.println(pagina);
    }
} catch (Exception e) {
    System.out.println("Error al procesar el fichero de favoritos: " + e.getMessage());
    e.printStackTrace();
}

```

Resultado de la ejecución del código anterior:

```

Output - ejemploSAX (run-single)
init:
deps-jar:
Created dir: D:\proyectosCursos\ejemploSAX\build
Updating property file: D:\proyectosCursos\ejemploSAX\build\build-jar.properties
Created dir: D:\proyectosCursos\ejemploSAX\build\classes
Created dir: D:\proyectosCursos\ejemploSAX\build\empty
Created dir: D:\proyectosCursos\ejemploSAX\build\generated-sources\ap-source-output
Compiling 1 source file to D:\proyectosCursos\ejemploSAX\build\classes
compile-single:
run-single:
Direccion de la pagina: http://www.javahispano.org valoracion: 10
Direccion de la pagina: http://www.gft.com valoracion: 1
BUILD SUCCESSFUL (total time: 1 second)

```

## 6.3. JAX-RPC

El Remote Procedure Call (RPC) es un protocolo que permite a un programa ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

JAX-RPC es un api basada en RPC que permite a una aplicación Java invocar un servicio web basado en Java con una descripción conocida sin dejar de ser consistente con su descripción WSDL. El servicio JAX-RPC utiliza estándares del W3C.

JAX-RPC facilita el uso de servicios web además permite el desarrollo de servicios web, especialmente si usamos la plataforma J2EE. Un servicio Web basado en RPC básicamente es una colección de procedimientos que pueden ser llamados por un cliente remoto desde Internet.

Muestra de invocación de un servicio web:



*Comunicación con Servicio Web<sup>12</sup>*

Definición de una Interface remota:

```
package helloservice;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```

Implementación de la interface remota:

```
package helloservice;

public class HelloImpl implements HelloIF {

    public String message ="Hello";

    public String sayHello(String s) {
        return message + s;
    }
}
```

<sup>12</sup> <http://docs.oracle.com/javaee/1.4/tutorial/doc/images/jaxrpc-myHelloServiceApp.gif>

Configuración del servicio:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
    name="MyHelloService"
    targetNamespace="urn:Foo"
    typeNamespace="urn:Foo"
    packageName="helloservice">
    <interface name="helloservice.HelloIF"/>
  </service>
</configuration>
```

Clase cliente de servicio web:

```
package staticstub;

import javax.xml.rpc.Stub;

public class HelloClient {

    private String endpointAddress;

    public static void main(String[] args) {

        System.out.println("Endpoint address = " + args[0]);
        try {
            Stub stub = createProxy();
            stub._setProperty
                (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
                 args[0]);
            HelloIF hello = (HelloIF)stub;
            System.out.println(hello.sayHello("Duke!"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private static Stub createProxy() {
        // Note: MyHelloService_Impl is implementation-specific.
        return
```

```
        (Stub) (new MyHelloService_Impl().getHelloIFPort());  
    }  
}
```

Al ejecutar el cliente del servicio web, se obtiene:

**Hello Duke!**

# MESOGRAFÍA

- XML. Miguel Ángel Acera García. Editorial Anaya Multimedia, 2012  
ISBN 9788441529601
- Beginning XML, 5<sup>th</sup> Edition. Joe Fawcett, Liam Quin. Editorial Wrox Press, 2012  
ISBN 978-1-118-16213-2
- Joe Fawcett, Liam Quin . Beginning XML, Editorial Wrox Press, Edición 2012
- Implementación de JAX-RPC en <https://java.net/projects/jax-rpc>.  
Fecha de consulta: 15 de Febrero de 2014
- API de Java para Procesamiento de XML, JCP en <https://www.jcp.org/en/jsr/detail?id=206>.  
Fecha de consulta: 17 de Febrero de 2014
- Java y XML: JDOM en <http://www.latascadexela.es/2008/08/java-y-xml-jdom.html>.  
Fecha de consulta: 17 de Febrero de 2014
- Tecnologías XML en <http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML>.  
Fecha de consulta: 20 de Febrero de 2014
- World Wide Consortium en <http://www.w3.org/>.  
Fecha de consulta: 22 de Febrero de 2014

## **UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**DR. JOSÉ NARRO ROBLES**  
Rector

**DR. EDUARDO BÁRZANA GARCÍA**  
Secretario General

**ING. LEOPOLDO SILVA GUTIÉRREZ**  
Secretario Administrativo

**DR. FRANCISCO JOSÉ TRIGO TAVERA**  
Secretario de Desarrollo Institucional

**LIC. ENRIQUE BALP DÍAZ**  
Secretario de Servicios a la Comunidad

**LIC. LUIS RAÚL GONZÁLEZ PÉREZ**  
Abogado General

**RENATO DÁVALOS LÓPEZ**  
Director General de Comunicación Social

## **DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN**

**DR. FELIPE BRACHO CARPIZO**  
Director General

**I.Q. ADELA CASTILLEJOS SALAZAR**  
Directora de Docencia en Tecnologías de Información y Comunicación

**MTRO. JESÚS DÍAZ BARRIGA ARCEO**  
Coordinador de Planeación Académica

**MTRA. ALEJANDRINA SAN JUAN REYES**  
Coordinadora del Centro Nuevo León de Extensión Académica  
en Tecnologías de Información y Comunicación

**ING. SERGIO ALVA ARGUINZONIZ**  
Coordinador del Centro Mascarones de Extensión Académica  
en Tecnologías de Información y Comunicación

**LIC. PATRICIA ROMERO ELÍAS**  
Coordinadora del Centro Polanco de Extensión Académica  
en Tecnologías de Información y Comunicación

**LIC. JOSÉ LUIS JANITZIO MEDINA FLORES**  
Coordinador del Centro San Agustín de Extensión Académica  
en Tecnologías de Información y Comunicación

**ING. PABLO DE LA O CRUZ**  
Coordinador del Centro Tlatelolco de Extensión Académica  
en Tecnologías de Información y Comunicación

*Junio, 2014*

**Diplomado Desarrollo de Sistemas con  
Tecnología Java**  
**Módulo 5: Uso de XML en la creación de interfaces  
web e intercambio de información**

Fue editado e impreso por la Dirección General de Cómputo y  
de Tecnologías de Información y Comunicación, Circuito  
Exterior, Ciudad Universitaria, Coyoacán, C.P. 04510,  
México DF., Junio de 2014

El tiraje consta de 100 ejemplares.

Se utilizaron tipos Arial de 13/17, 16/22, 26/42 pts. Tahoma  
8/10, 9/15, 11/13 de pts. y discos compactos de 700 MB de  
capacidad.