XML

Extensible Markup Language

INTRODUCCIÓN HISTÓRICA

XML es un lenguaje libre y abierto (al igual que HTML y CSS).

Historia

En los años ’60 los documentos incluían códigos de control para definir la presentación del texto: ya se marcaba la información, pero con formatos propietarios, cerrados y dependientes del programa que procesaba.

A finales de los ’60 empezaba a surgir la necesidad de un formato estándar y común para publicaciones que separase el contenido y la forma en que este se va a presentar. Y en 1971 surge GML (Generalized Markup Language) de IBM, un lenguaje extensible y legible, con el que llega la necesidad de crear mecanismos de validación.

ANSI estandariza GML en 1980-84 e ISO en 1986 (en colaboración con ANSI), formalizando el concepto de DTD (un documento separado del documento cuya estructura se especifica).

SGML se centra en la relación estructura-contenido y separa estructura, contenido y estilo. Formaliza el concepto de DTD, un documento separado pero relacionado con el documento de contenido y que especifica la estructura de este. SGML, sin embargo, era un estándar muy pesado y complejo (aumenta el precio, el consumo de recursos y reduce los navegadores que lo soportan), por esto ha caído en favor de XML.

Tanto XML como HTML derivan de SGML (Standarized GML).

Así nació HTML, con el propósito de facilitar mecanismos independientes de plataforma para documentos de hipertexto, con enlaces entre ellos, ya sea en el mismo servidor, red de área local o Internet.

HTML fue una gran ayuda para la difusión de contenidos de forma sencilla y rápida vía Internet. Al ser gratuito, muy simple y estar ampliamente soportado se hizo muy popular (pues permitía a cualquiera tener una página web). Pero tiene inconvenientes como que no se puede extender, no hay estructura semántica y las empresas añaden etiquetas propietarias.

Otras tendencias son las aplicaciones DTP (TEX, LaTEX) orientadas a la presentación. (DTP: software destinado a la autopublicación que combina texto editable con imágenes -ejemplos son PageMaker, QuarkXPress, Adobe InDesign, RagTime, Scribus, MS Publisher, Corel Ventura, Apple Pages).

XML

En 1996 el W3C (World Wide Web Consortium) patrocina a un grupo de expertos en SGML para que desarrollen un lenguaje de marcas con la potencia y extensibilidad de SGML pero la simplicidad de HTML. Eliminaron todo lo que no era básico o no se estuviera usando para hacer un lenguaje más fácil y simple de implementar, orientado a la web. Así nace XML 1.0 alrededor de 1998.

XML es un subconjunto de SGML que tiene el fin de permitir que SGML genérico se envíe, se reciba y se procese en internet de la misma forma que ocurre con HTML. Ha sido diseñado para una implementación sencilla y para tener la capacidad de trabajar con SGML y HTML sin necesidad de configuraciones adicionales.

XML es un lenguaje de marcas o metalenguaje. Un metalenguaje es un lenguaje usado para describir formalmente otro lenguaje, o lo que es lo mismo, crear vocabularios (por ejemplo, un lenguaje para representar fórmulas matemáticas) -estas descripciones o vocabularios se pueden llamar aplicaciones de XML. Así, XML actúa como estándar según el cual los desarrolladores pueden crear lenguajes específicos con sus propias etiquetas, estructuras, etc.

Una marca es una forma de añadir información sobre los datos a los propios datos: insertamos símbolos o caracteres en un documento para indicar su estructura lógica o física (cómo la información se debería presentar). El objetivo principal es separar el tratamiento (ej. la apariencia) del documento de los datos.

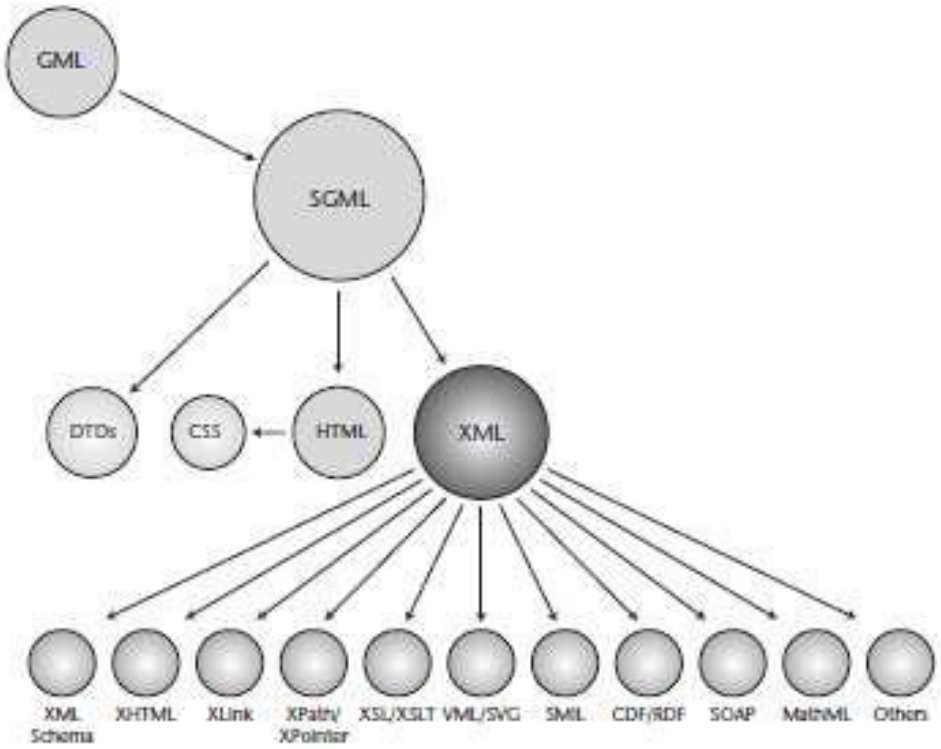
XML no limita las marcas que se van a crear, solo el formato (deben estar basadas en caracteres y ser legibles).

Es independiente del software y de la plataforma (un mismo documento XML puede ser usado en cualquier sistema operativo con cualquier herramienta de desarrollo o publicación). Es un formato público (un estándar abierto) que no depende de empresas ni países. Existen herramientas gratuitas para trabajar con él (gracias a su facilidad de implementación).

Una de las ventajas de XML es que el diseñador se puede centrar en el uso de etiquetas, el contenido y su estructura (formatear el contenido es un proceso posterior o incluso secundario). El proceso de validación comprueba la sintaxis (si está bien formado) y la estructura.

Sobre sus similitudes con HTML y SGML: XML es una versión reducida de SGML (no un HTML ampliado), pues se diseñó pensando en la estructura y el significado de los datos y HTML se creó pensando en la forma en que se presentan los datos.

Las etiquetas XML describen datos. Su sintaxis requiere precisión y seguir estrictamente las reglas gramaticales.



INTRODUCCIÓN A XML

XML significa Extensible Markup Language.

Fue creado para organizar, almacenar y transportar información.

Las etiquetas XML (tags) no están predefinidas (al contrario de lo que ocurre en HTML, que solo se pueden utilizar las estandarizadas o definidas por el navegador): permite definir nuestras propias etiquetas y la estructura del documento. Hay que tener en cuenta que esto obliga a la comunicación del formato a terceros que vayan a trabajar con nuestros documentos.

Un documento XML no hace nada.

XML está diseñado para ser autodescriptivo.

La codificación por defecto es UTF-8. Debemos comprobar la codificación de nuestro editor para evitar conflictos.

ESTRUCTURA Y REGLAS

Los documentos XML deben contener un elemento raíz (root) que será el padre de todos los demás en la estructura en árbol.

Los elementos pueden tener contenido textual, atributos y otros elementos que se convierten en contenedores.

· Los comentarios se utilizan para legibilidad o para deshabilitar secciones y se realizan así: <!— Comentario -->

· Los espacios en blanco no se truncan (aparecen los mismos que se han escrito).

· Las nuevas líneas se almacenan como “lf” (*line feed*), al contrario de lo que hace Windows (cr + lf)

· Las secciones CDATA se utiliza para evitar que ciertos caracteres se ejecuten: le indican al parser que ignore el marcado de la sección y pase esos caracteres como texto a la aplicación. Los delimitadores de sección CDATA son <![CDATA[(inicio) y]]> (fin). Se suelen utilizar para escapar un trozo de código. Si solo se quiere escapar un carácter se puede utilizar su código HTML (ej. <).

Reglas

· Deben tener un elemento raíz.

· Todos los elementos deben tener etiqueta de cierre o final.

· Se distinguen mayúsculas y minúsculas.

· Los elementos deben estar correctamente anidados.

· Los atributos deben estar entrecomillados (atributo="valor").

· Los nombres de los elementos pueden contener letras, números y otros caracteres. No pueden comenzar con número ni carácter de puntuación ni contener espacios. Se puede usar cualquier nombre (no hay palabras reservadas). Están prohibidos los símbolos “<” y “&”.

&lt;	< less than
&gt;	> greater than
&amp;	& ampersand
&apos;	' apostrophe
&quot;	" quotation mark

Buenas prácticas

· Los nombres deben ser significativos pero simples y breves. Se pueden utilizar guiones bajos (primer_apellido) o la notación camel (primerApellido). Es recomendable evitar guiones (“-“ se confunde con la resta), puntos (confusión con objetos y propiedades) y dobles puntos (“:”, reservados para los espacios de nombrado). Se recomienda evitar caracteres externos al alfabeto anglosajón (podrían generar problemas en el software de terceros). Si el documento viene de una tabla o base de datos, se conservarán los nombres de los datos.

· Si los XML se corresponden con una tabla o base de datos, se deberían seguir las reglas de nombrado de la base de datos en el documento XML.

· Es mejor evitar los nombres de elementos que solo difieren en una letra (serie vs series) porque esto aumenta la probabilidad de error.

· Si tengo dos elementos con el mismo nombre con dos estructuras distintas no puedo utilizarlos en mi dtd.

· Se debe evitar, si es posible, tener elementos o contenido mixto. Esto significa que dentro de un elemento haya contenido (ej. 10) y otros elementos. Ejemplos:

```
<contacto>Ana Pérez
        <tel>123</tel>
        <mail>a@a.com</mail>
</contacto>
```

Esto lo que hace luego es complicarme la vida cuando llega la fase de validación. Para evitarlo simplemente se crea un subelemento nuevo (en este caso podría ser "nombre").

Si necesitamos dos elementos del mismo nombre ("nombre") o bien lo definimos una vez y solo puede ser de un tipo de dato (el definido) o bien tenemos que cambiar el diseño (los nombres de los elementos) o abandonar las DTD y utilizar un esquema.

ATRIBUTOS

Los atributos proporcionan información adicional sobre el elemento. Deben ir siempre asociados a un elemento (se definen dentro de la etiqueta de inicio del elemento) y cada elemento puede tener más de uno.

Los valores de los atributos deben estar entrecomillados, con comilla simple o doble: si el atributo contiene comillas dobles se usarán las simples para acotarlo (o bien se sustituirán por los caracteres predefinidos para evitar la introducción directa de las comillas).

```
<gangster name='George "Shotgun" Ziegler'>
<gangster name="George &quot;Shotgun&quot; Ziegler'>
```

Los atributos se declaran así:

```
<book category="CHILDREN"> <title lang="en">Harry Potter</title> </book>
```

Es recomendable no abusar de los atributos y utilizar elementos en su lugar: se deben usar elementos para los datos, atributos para la información que no es relevante para los datos (es decir, los metadatos o datos sobre los datos).

Los inconvenientes de los atributos son que no pueden contener valores múltiples, estructuras en árbol ni se pueden expadir o extender con facilidad en el futuro (ej. la fecha puedo querer desglosarla en día, mes y año en un futuro, y de ponerla como un atributo no podría), además de ser difíciles de leer y mantener.

Se recomienda usar elementos para los datos y atributos para información que no sea relevante para los datos (es decir, metadatos: datos sobre los datos).

VISIÓN DE NAVEGADOR

Los distintos navegadores y las distintas versiones de estos nos suelen presentar el documento XML de forma similar: con los elementos coloreados y símbolos para contraer y expandir la estructura (no todos muestran la declaración de fichero xml). La extensión del fichero debe ser .xml (si no, el navegador lo podrá interpretar como un fichero de texto u otro).

Para ver el código fuente del documento hay que ir a View Page Source o View Source. Aun así, en algunos navegadores solo se muestran los elementos de texto: si usamos el inspector de elementos (F12 en Firefox) vemos la transformación por defecto que hace el navegador a HTML y CSS para presentarlo.

Si hay errores, el navegador mostrará un mensaje de error. (Aunque algunos lo ocultan en la consola.)

Los documentos XML no llevan información sobre su presentación, así que se presentan tal cual. Algunas soluciones que se pueden aplicar para su presentación son hojas de estilo (CSS), lenguajes de transformación (XSLT) o JavaScript.

Al abrir uno de estos ficheros se abre el navegador automáticamente. La visualización cambia un poco: en IE sale al principio el elemento <?xml version="1.0"?> y en Firefox "This XML file does not appear to..." (no se encuentra la plantilla de configuración o algo así).

DECLARACIONES INICIALES

Un fichero XML tiene dos partes: el prólogo y la instancia de datos.

PRÓLOGO

La primera línea de nuestro documento XML es normalmente:

<?xml version=”1.0” encoding=”UTF-8”?>

Es obligatorio incluir la versión.

La codificación por defecto es UTF-8 (hay que comprobar la codificación del editor para que sea la misma), y para definirla utilizamos el atributo encoding, aunque no es obligatorio (al igual que el atributo *standalone=”yes|no”* que ¿dice si hay un DTD externa?).

La declaración de tipo de documento (interno/externo) que se usa para validar y su localización se indican mediante DOCTYPE (donde “note” es el nombre del elemento raíz del XML y “note.dtd” el nombre del fichero contenedor de la DTD). La palabra “system” es opcional: *system* se utiliza para dtds privadas (mías) y *public* para dtds estándar (ya existentes).

<!DOCTYPE note SYSTEM “note.dtd”>

Instrucciones de proceso (IP), destinadas a la aplicación (por ejemplo, presentación):

<?xml-stylesheet type=”text/css” href=”cd_catalog.css” ?>

INSTANCIA DE DATOS

Tras estas primeras líneas iniciales abrá una estructura de árbol: un único elemento o nodo raíz (contenedor) dentro del que estarán los demás (hijos, etc.).

```
<root>
        <child>
                <subchild>...</subchild>
        </child>
</root>
```

Ejemplo:

```
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

VALIDACIÓN

Se comprueba que el documento esté:

· Bien formado (con una sintaxis correcta), para esto se comprueba que siga 5 reglas: raíz, etiquetas de cierre, mayúsculas/minúsculas, anidamiento y atributos entre comillas.

· Validado (contra una DTD o un schema): está bien formado, incluye una referencia a una DTD o schema donde se especifique la estructura del documento XML y cumple lo especificado en esa DTD o schema.

Los errores en la sintaxis detienen el procesamiento del documento.

Una forma de validar (la más habitual) es utilizar DTD. Para cosas más complejas utilizaremos los ficheros XML y otras cosas.

Los navegadores no validan, pues no tienen en cuenta los documentos externos al XML (como la DTD).

DTD (DOCUMENT TYPE DEFINITION)

La DTD es una plantilla que especifica las normas que debe seguir un documento XML concreto: define la estructura de un documento XML (la lista de elementos, atributos válidos, etc.). Esto resulta útil a la hora de intercambiar datos entre diferentes grupos.

Puede ser declarada dentro del documento, como una referencia externa o ambas.

DTD externa

- Fichero XML: referencia a DTD externa en DOCTYPE:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- Fichero DTD: fichero "**note.dtd**" con la DTD sería:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

DTD interna

- Sólo fichero XML:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

Puede ser privada o de acceso público: esto se declara en la DOCTYPE. Las DTD privadas pueden estar almacenadas en local (<!DOCTYPE tablon SYSTEM "tablon.dtd">) o en una web (<!DOCTYPE tablon SYSTEM "http://www.ae.es/tablon.dtd">) -en este caso la DTD es pública pero la empresa tiene control sobre ella. Cuando la DTD se considera un estándar para uso público, se declara como DTD de acceso público (<!DOCTYPE tablon PUBLIC fpi URL>); el fpi es el identificador público formal.

DECLARACIONES DE LA DTD

La sintaxis de la dtd es muy importante, incluidos espacios: si quito el espacio entre el nombre del elemento y el paréntesis de apertura, da error. Ej. <!ELEMENT contacto(#PCDATA)>

<!ELEMENT note (message)> indica que el elemento hijo "message" aparecerá una vez.

<!ELEMENT note (to, from, heading, body)>

La coma (**,**) indica una secuencia obligatoria de elementos.

?	0-1
*****	0-n
+	1-n

El **+** indica que el elemento aparecerá al menos una vez. <!ELEMENT tablon (aviso+) >

El ***** indica que el elemento aparecerá cero o más veces. <!ELEMENT clase (alumno*)>

La **?** indica que el elemento aparecerá cero o una vez. <!ELEMENT aviso (msg, priority?)>

La **|** (pipe) indica que solo aparecerá uno de los elementos indicados. <!ELEMENT note (to, from, (message | body))>

<!ELEMENT note (message)> ---> tiene que haber un elemento sí o sí (1)

<!ELEMENT tablon (aviso+)> --> habrá al menos un elemento (1 o más)

<!ELEMENT clase (alumno*)> --> puede haber o no uno o varios elementos (0 o más)

<!ELEMENT aviso (priority?)> --> puede aparecer una vez o ninguna (0-1)

<!ELEMENT note (to, from, (message | body))> --> aparece 1 to, 1 from y 1 (message o body)

Tipos de elementos

· **PCDATA** (*parsed character data*): texto que será procesado y analizado por el parser en busca de entidades o etiquetas (no debería contenter &, <, > sino sus equivalentes & < y >).

<ELEMENT from (#PCDATA)>

· **CDATA** (character data): texto que no será analizado por el parser.

· **EMPTY** (elemento vacío): no tiene texto, pero puede tener atributos. <!ELEMENT vacio EMPTY>

· **ANY**: no se comprueba nada. <!ELEMENT batiburrillo ANY>

Si tengo un elemento o contenido mixto, para conseguir que la DTD me lo valide tendré que poner algo así: <!ELEMENT estadoInv (#PCDATA | orderMsg)> para el siguiente XML. (Debe existir un espacio en blanco antes y después de la barra vertical.)

```
<estadoInv> 10
        <orderMsg> pedido impresora</orderMsg>
</estadoInv>
```

#PCDATA puede tener elementos html (se utiliza para elementos)

CDATA significa datos de tipo carácter (se utiliza para atributos)

ATRIBUTOS

<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorpordefecto>

<!ATTLIST pago tipo CDATA “contado”> (DTD)

<pago tipo=“tarjeta”>ordenador</pago> (XML)

También sería válido <pago />.

El valor por defecto se puede especificar en la dtd pero luego las herramientas tienen que saber que lo tienen que usar (por ejemplo, como el navegador no mira la dtd, no muestra el valor por defecto).

ATTLIST sirve para definir una lista de atributos (no hay que poner un attlist para cada atributo).

<!ATTLIST anuncio vendido CDATA #REQUIRED

codigo ID #REQUIRED >

Se puede requerir que el valor de un atributo sea obligatorio, fijo u otros:

#REQUIRED (el atributo es obligatorio)

<!ATTLIST alumno numero CDATA #REQUIRED>

Esto será un atributo "numero" que pertenece al elemento "anuncio" y es obligatorio (#REQUIRED).

#IMPLIED (el atributo es opcional y no tiene valor por defecto)

<!ATTLIST contacto fax CDATA #IMPLIED> -> <contacto /> en el XML sería válido

#FIXED "valor" (no se puede cambiar, siempre debe llevar ese valor el atributo)

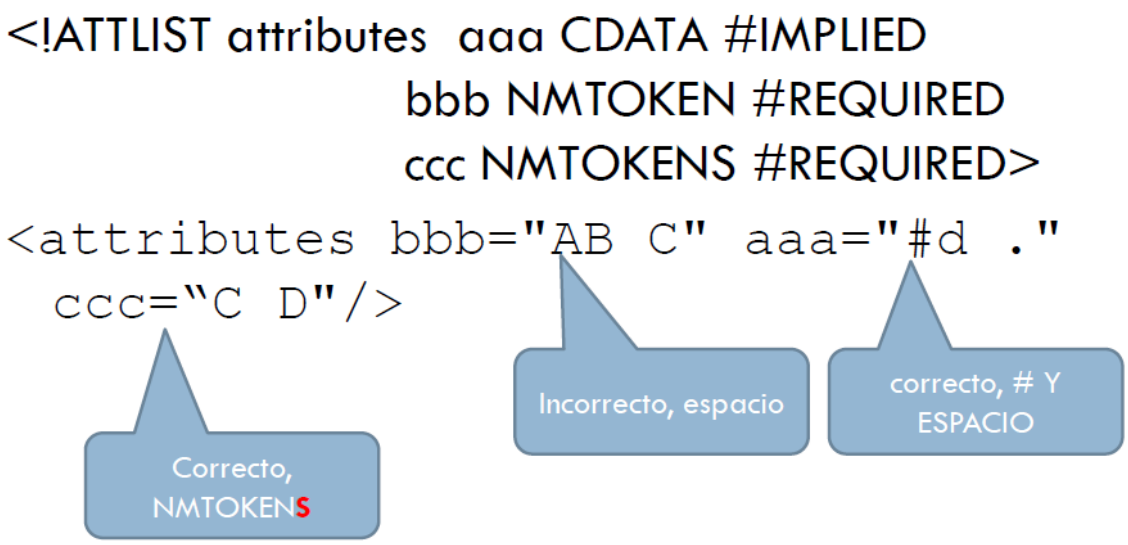
<!ATTLIST envio empresa CDATA #FIXED "MS">

Tipos de atributos

· **CDATA**: puede contener cualquier carácter si este se atiene a las reglas de formación.

· **NMTOKEN**: solo puede contener letras, dígitos, punto, guión, subrayado y dos puntos.

· **NMTOKENS**: puede contener los mismos caracteres que NMTOKEN más espacios en blanco. Un espacio en blanco consiste en uno o más espacios, retornos de carro o tabuladores. Permite poner más de un valor en el mismo atributo.



· **ID**: el valor de un atributo de tipo ID solo puede contener los mismos caracteres que un NMTOKEN y debe empezar por letra. Ningún tipo de elemento puede tener especificado más

de un atributo de tipo ID. El valor de un atributo ID debe ser único entre todos los valores de atributos ID.

· **IDREF**: su valor debe corresponder con el valor de algún atributo ID del documento.

Si cada alumno tiene un id del profesor: será un idref (que apunta al id del profesor).

· **IDREFS**: puede contener varias referencias a elementos con atributos ID separados por espacios en blanco.

· **Enumerado**: un valor entre un conjunto, (valor1 | valor2 …), puede llevar REQUIRED o IMPLIED. El enumerado puede venir bien para cuando hay 2-4 opciones. (Debe existir un espacio en blanco antes y después de la barra vertical.)

<!ATTLIST pago tipo (contado | CC) “contado” #REQUIRED>

<!ATTLIST contacto visible (si | no) #IMPLIED >

<!ATTLIST contacto visible (si | no) “no” >

<!ATTLIST f1 pais (ESP | FRA | ITA | ALE) "ESP">

Así solo puede rellenarse el atributo descarga con las opciones "directa" o "emule":

<!ATTLIST enlace descarga (directa | emule) #REQUIRED>

Así podría meterse cualquier dato:

<!ATTLIST enlace descarga CDATA #REQUIRED>

ENTIDADES

Se pueden crear variables que actúan como atajos o abreviaturas:

[DTD] <!ENTITY writer “Donald Duck”>

[XML] <author>&writer;</author>

Estas variables pueden apuntar a una url externa. La ventaja de esto es que se pueden compartir con muchos documentos XML y solo hay que actualizar una localización si es necesario.

<!ENTITY writer SYSTEM “http://www.w3schools.com/entities.dtd”>

Los navegadores pueden dar error con este tipo de entidades.

CSS*: HOJAS DE ESTILO*

Hay tres posibilidades para la presentación del XML: CSS, XSLT y lenguajes de programación (JavaScript, PHP…).

El documento CSS utilizará los nombres (y clases) de los elementos XML para aplicarles estilos.

Los elementos de un xml en principio son “in line”.

Para conectar el css con el xml tendré que poner en este último lo siguiente:

<?xml version="1.0" ?>

<?xml-stylesheet type="text/css" href="cd_catalog.css"?>

A partir de este XML:

```
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
```

El CSS tendrá este aspecto:

Fichero: cd_catalog.css

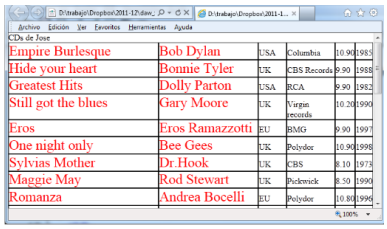
```
CATALOG {
background-color: #ffffff;
width: 100%
}
CD {
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE {
color: #FF0000;
font-size: 20pt;
}
ARTIST{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY{
display: block;
color: #000000;
margin-left: 20pt;
}
```

Etiquetas elementos

Al abrir el XML en el navegador veré rápidamente si tengo un problema con el CSS.

Si quiero presentar el XML como una tabla solo tengo que tener claro qué es la tabla, qué son las filas y qué son las celdas.

```
CATALOG {
 display: table
}
CD {
 display: table-row;
}
TITLE, ARTIST, .. {
 display: table-cell;
 border: 1px solid black;
}
```



CDs de Jose

Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1983
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988
Greatest Hits	Dolly Parton	USA	RCA	9.90	1982
Still got the blues	Gary Moore	UK	Virgin records	10.20	1990
Eros	Eros Ramazzotti	EU	BMG	9.90	1997
One night only	Bee Gees	UK	Polydor	10.90	1998
Sylvias Mother	Dr.Hook	UK	CBS	8.10	1973
Maggie May	Rod Stewart	UK	Pickwick	8.50	1990
Romanza	Andrea Bocelli	EU	Polydor	10.80	1996

XSLT (Extensible Stylesheet Language Transformations)

Se puede usar para transformar XML (fuente) en HTML o en otro documento XML (resultado). XSL es mucho más potente que HTML y CSS porque te permite filtrar los resultados ordenados por diferentes cosas, y es muy flexible (con él puedo eliminar, añadir, ordenar información, realizar test, usar selectivas -if- y bucles).

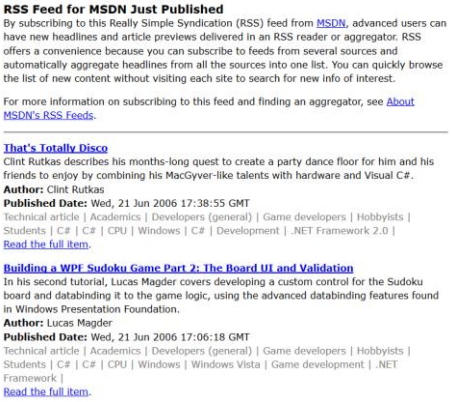
Para añadir estilos mediante CSS puedo utilizar el elemento *style* (dentro de *head*, en el *html*) o incluir una referencia al fichero de estilos en un elemento *link* (dentro de la cabecera del *html*).

Una hoja de estilos XSL es un documento XML, así que debe llever la declaración de XML al principio:

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

Ejemplo fichero XML sin XSLT y con XSLT:

```
- <rss version="2.0">
  - <channel>
      <title>MSDN Just Published</title>
      <link>http://msdn.microsoft.com/</link>
    - <description>
        Keep current with all the new technical articles, columns, specifications, and resources published on
      </description>
      <language>en-us</language>
      <pubDate>Thu, 22 Jun 2006 13:02:23 GMT</pubDate>
      <lastBuildDate>Thu, 22 Jun 2006 13:02:23 GMT</lastBuildDate>
      <generator>MSDN and TechNet RSS Service 1.1.0.0</generator>
      <ttl>1440</ttl>
    - <item>
        <title>That's Totally Disco</title>
      - <description>
          Clint Rutkas describes his months-long quest to create a party dance floor for him and his friends
        </description>
      - <link>
          http://msdn.microsoft.com/coding4fun/coolapplications/disco/default.aspx
        </link>
        <dc:creator>Clint Rutkas</dc:creator>
        <category domain="mscomdomain:ContentType">Technical article</category>
        <category domain="mscomdomain:Audience">Academics</category>
        <category domain="mscomdomain:Audience">Developers (general)</category>
        <category domain="mscomdomain:Audience">Game developers</category>
```



RSS Feed for MSDN Just Published

By subscribing to this Really Simple Syndication (RSS) feed from MSDN, advanced users can have new headlines and article previews delivered in an RSS reader or aggregator. RSS offers a convenience because you can subscribe to feeds from several sources and automatically aggregate headlines from all the sources into one list. You can quickly browse the list of new content without visiting each site to search for new info of interest.

For more information on subscribing to this feed and finding an aggregator, see About MSDN's RSS Feeds.

That's Totally Disco
Clint Rutkas describes his months-long quest to create a party dance floor for him and his friends to enjoy by combining his MacGyver-like talents with hardware and Visual C#.
Author: Clint Rutkas
Published Date: Wed, 21 Jun 2006 17:38:55 GMT
Technical article | Academics | Developers (general) | Game developers | Hobbyists | Students | C# | C# | CPU | Windows | C# | Development | .NET Framework 2.0 | Read the full item.

Building a WPF Sudoku Game Part 2: The Board UI and Validation
In his second tutorial, Lucas Magder covers developing a custom control for the Sudoku board and databinding it to the game logic, using the advanced databinding features found in Windows Presentation Foundation.
Author: Lucas Magder
Published Date: Wed, 21 Jun 2006 17:06:18 GMT
Technical article | Academics | Developers (general) | Game developers | Hobbyists | Students | C# | C# | CPU | Windows | Windows Vista | Game development | .NET Framework |
Read the full item.

La transformación se puede hacer en cliente (navegador) o en servidor (JS, PHP).

Opciones para transformar un xml utilizando un xsl:

1) Línea de comandos

2) XML Copy Editor (F8)

3) Abrir con el navegador

Con XSLT se pueden añadir o eliminar elementos y atributos, así como ordenar, realizar tests, tomar decisiones sobre qué elementos ocultar o mostrar.

Lo primero que debemos hacer es incluir la siguiente declaración en el XML, que enlazará el xsl:

<?xml-stylesheet type='text/xsl' href= 'fichero.xsl'?>

Este archivo .xsl tendrá la misma estructura que un archivo xml, solo que incluye algunos elementos extra (además de introducir HTML). Una hoja de estilos XSL consiste en uno o más conjuntos de reglas llamadas plantillas, que contienen reglas que se aplican cuando un nodo coincide con un patrón.

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/RAIZ">

<html>… </html>

<xsl:for-each select="CD"> <xsl:value-of select="TITLE"/> </xsl:for-each>

</xsl:template> </xsl:stylesheet>

Y ahora incluimos la declaración de documento XSL (en el XSL). El elemento *xsl:stylesheet* define el documento como una hoja de estilo XSLT e indica la versión (obligatorio) y el espacio de nombres XSLT que se utilizará para acceder a los elementos, atributos y características de XSL. Cualquiera de las dos líneas siguientes es igualmente válida.

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

o

<xsl:transform version="1.0" xmlns:xsl=" http://www.w3.org/1999/XSL/Transform">

Si pusiéramos en esta línea xmlns:xslt, con cambiar el prefijo de las etiquetas de xsl: a xslt: seguiría funcionando porque he definido este prefijo como etiqueta de transformación.

Plantillas de transformación

Plantillas que se aplican cuando coinciden con un nodo de mi fichero de datos.

Lo siguiente que es necesario introducir en nuestro documento XSL es el elemento **xsl:template**, que define una plantilla de transformación y la asocia a un elemento del XML fuente mediante el atributo **match** (match="/" la asocia a la raíz del XML: el elemento contenedor del elemento raíz). El contenido de la plantilla (del elemento plantilla) puede definir código HTML que formará parte del documento salida.

<xsl:template match="/"> dice "si encuentras el elemento barra, muestra lo que hay dentro de este elemento xsl:template.

Si tengo <xsl:template match="/CATALOG"> y dentro <xsl:value-of select="TITULO"> me buscará un título dentro del catálogo. Pero si mi primera match pone match="/" lo que hace es buscar la raíz, y dentro de la raíz (no dentro del catálogo) un título.

A veces se utiliza la raíz (/) en lugar del elemento raíz para que mi hoja de estilos se pueda aplicar a muchas xml.

Es dentro de este elemento donde introduciremos la información que queremos que aparezca en el documento final y las transformaciones del XML que han de realizarse para obtener el resultado final. Todo esto se consigue utilizando elementos HTML y elementos XSL.

ELEMENTOS XSL

Todas los elementos XSL llevan un prefijo que las diferencia del resto del código (xsl:).

<xsl:value-of select=""> se usa para extraer el valor del nodo seleccionado y añadirlo al documento de salida

<xsl:value-of select="title" /> de <xsl:value-of select="artist" />

<xsl:for-each select=""> crea un bucle sobre los elementos XML (se usa para seleccionar todos los elementos de un conjunto especificado de nodos del árbol XML.

<xsl:for-each select="catalog/cd[artist='Bob Dylan']">

El atributo **select** contiene una expresión Xpath (url relativa que lleva una arroba delante del atributo elegido). Dentro de este atributo se pueden incluir operadores de comparación (=, !=, < , >), como en cualquier Xpath.

<xsl:value-of select="@propietario" />

<xsl:value-of select="artist/@pais" />

xsl:sort ordena alfabéticamente (¡cuidado!, los números también: hay que decirle que el tipo de dato son números). Se suele introducir este elemento dentro de un for-each.

<xsl:sort select="price" data-type:"number" order="descending" />

--

```
<xsl:for-each select="catalog/cd">
    <xsl:sort select="artist"/>
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
  </xsl:for-each>
```

--

El siguiente código me ordenará por precio descendente y cuando haya precios iguales por artista alfabéticamente:

```
<xsl:sort select="PRICE" order="descending" data-type="number" />
<xsl:sort select= "ARTIST" />
```

Para que solo me salgan los discos de un artista:

```
<xsl:for-each select="CD[ARTIST='Bob Dylan']>
```

O los que no son de un artista:

```
<xsl:for-each select="CD[ARTIST!='Bob Dylan']>
```

O los que salieron después de 1982 (para esto deberé poner el '>' como un código: >):

```
<xsl:for-each select="CD[YEAR &gt; '1982']">
<xsl:sort select="ARTIST" />
```

Cambia el estilo de los discos más caros:

```
<xsl:choose>
<xsl:when test=”PRICE&gt;10”>
<tr>
<td><xsl:value-of select=”TITLE” /></td>
<td><xsl:value-of select=”ARTIST” /></td>
<td class=”caro”> <xsl:value-of select=”PRICE” /> </td> (a la clase “caro” le puedo
haber puesto color de fondo rojo por ejemplo)
</tr>
</xsl:when>
<xsl:otherwise>
<tr>
<td><xsl:value-of select=”TITLE” /></td>
<td><xsl:value-of select=”ARTIST” /></td>
<td> <xsl:value-of select=”PRICE” /> </td>
</tr>
</xsl:otherwise>
</xsl:choose>
</xsl:for-each>
```

<xsl:if test=“expresión**”>** produce una salida si la expresión es verdadera; se suele añadir dentro del elemento xsl:for-each.

```
<xsl:for-each select="catalog/cd">
    <xsl:if test="price &gt; 10">
```

El valor del atributo “test” contiene la expresión que será evaluada.

xsl:choose es una selectiva/condicional múltiple (el equivalente a un switch en C#). Se usa en conjunto con **xsl:when** (para cuando se cumpla la condición indicada) y **xsl:otherwise** (si no se cumple ninguna de las anteriores).

```
<xsl:choose>

<xsl:when test=“price &gt; 10”> … </xsl:when>

<xsl:when test=“expresión”> … </xsl:when>

…

<xsl:otherwise> … </xsl:otherwise>

</xsl:choose>
```

EXPRESIONES XPATH

Los documentos XML son tratados como árboles de nodos: el superior es el raíz (*root element node*). Hay 7 tipos de nodos: elemento, atributo, texto, namespace, instrucción de proceso, comentario y nodo de documento. Los valores atómicos (*atomic values*) son nodos sin hijos ni padres (lo que vienen a ser los valores guardados en un elemento o un atributo).

Xpath utiliza expresiones para seleccionar nodos de un documento XML. Tiene más de 200 funciones integradas. Funcionan como las url relativas para navegar por el sistema de archivos:

/ selecciona un subdirectorio (si la expresión comienza con él, empieza en la raíz del XML)

@ selecciona un atributo

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Los predicados se usan para encontrar un nodo específico con cierto valor. Van encerrados en corchetes.

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: *In JavaScript:* *xml*.setProperty("SelectionLanguage","XPath");
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

También se puede utilizar comodines (*wildcards*) para seleccionar nodos desconocidos:

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

ejs.

Path Expression	Result
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

El uso del operador | permite elegir varios tipos de datos:

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

Un axis define un conjunto de nodos relativos al nodo actual:

AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Cada paso (/paso/paso) consiste en:

- an axis (defines the tree-relationship between the selected nodes and the current node)
- a node-test (identifies a node within an axis)
- zero or more predicates (to further refine the selected node-set)

La sintaxis de la localización de un paso es: **axisname::nodetest[predicate]**

Example	Result
child::book	Selects all book nodes that are children of the current node
attribute::lang	Selects the lang attribute of the current node
child::*	Selects all element children of the current node
attribute::*	Selects all attributes of the current node
child::text()	Selects all text node children of the current node
child::node()	Selects all children of the current node
descendant::book	Selects all book descendants of the current node
ancestor::book	Selects all book ancestors of the current node
ancestor-or-self::book	Selects all book ancestors of the current node - and the current as well if it is a book node
child::*/child::price	Selects all price grandchildren of the current node

Operadores:

Operator	Description	Example
|	Computes two node-sets	//book | //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80
or	or	price=9.80 or price=9.70
and	and	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

USOS

Ejemplos de uso de XML:

· SVG (tutorial en w3schools): los ficheros svg utilizan código XML

· Las listas de reproducción de música Wimpy player están escritas en XML.

· RSS (sindicación): el gestor de contenidos publica su información en formato RSS, que es XML (un tipo de XML llamado Atom).

· VMWare, VirtualBox, Microsoft y otros guardan la configuración de sus ficheros en .xml

BIBLIOGRAFÍA

· http://w3schools.com/xml/xml_whatis.asp (introducción) - Qué es un documento XML bien formado (well formed) y uno válido

· http://w3schools.com/xml/xml_validator.asp (validador)

· http://w3schools.com/xml/xml_view.asp

http://w3schools.com/xml/cd_catalog.xml

http://w3schools.com/xml/plant_catalog.xml

http://w3schools.com/xml/simple.xml