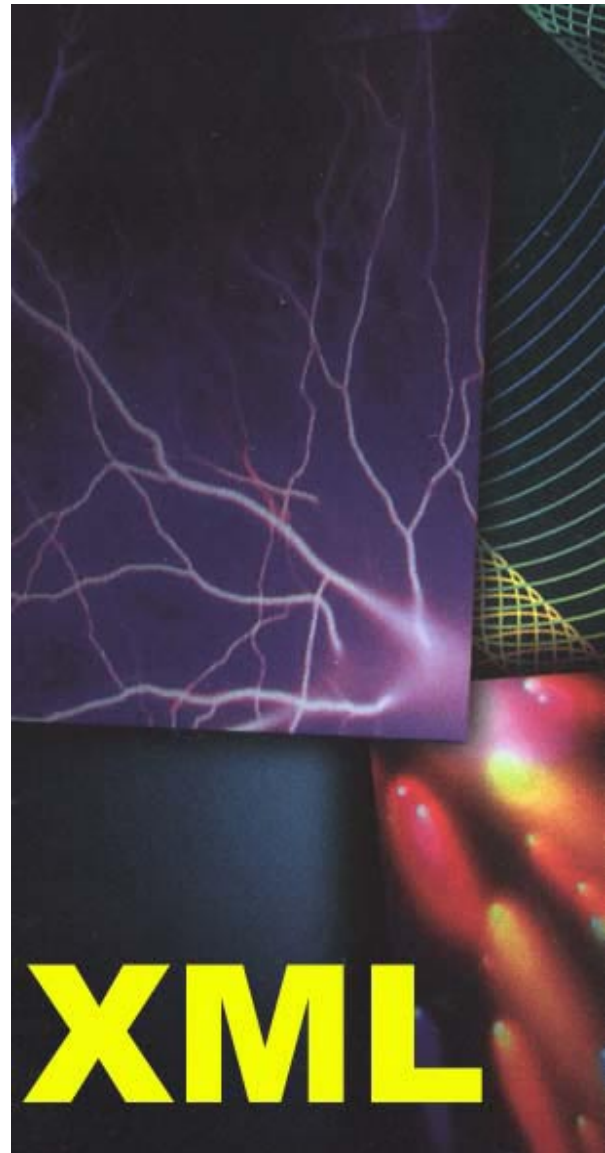


Este libro pretende ser una obra de carácter práctico, que permita al lector tener un buen conocimiento de todas las tecnologías que se ven implicadas dentro de lo que genéricamente se denomina XML, de forma que se consigan unas bases sólidas sobre las que trabajar, y se tengan claros cuáles son los patrones-guía por los que debe regirse y dónde buscar cuando las cosas se tuercen.

Respecto a los conocimientos previos, es importante que el lector maneje, siquiera básicamente, los fundamentos del lenguaje HTML, y para la parte final, donde se aborda el trabajo con modelos de objetos, conocimientos de algún lenguaje de programación que utilice estos modelos: Visual Basic, Delphi, Visual C++, Java, JavaScript, VBScript, Perl, etc.



INTRODUCCIÓN AL LENGUAJE MARINO POSADAS



ADVERTENCIA LEGAL

Todos los derechos de esta obra están reservados a Grupo EIDOS Consultoría y Documentación Informática, S.L.

El editor prohíbe cualquier tipo de fijación, reproducción, transformación, distribución, ya sea mediante venta y/o alquiler y/o préstamo y/o cualquier otra forma de cesión de uso, y/o comunicación pública de la misma, total o parcialmente, por cualquier sistema o en cualquier soporte, ya sea por fotocopia, medio mecánico o electrónico, incluido el tratamiento informático de la misma, en cualquier lugar del universo.

El almacenamiento o archivo de esta obra en un ordenador diferente al inicial está expresamente prohibido, así como cualquier otra forma de descarga (downloading), transmisión o puesta a disposición (aún en sistema streaming).

La vulneración de cualesquiera de estos derechos podrá ser considerada como una actividad penal tipificada en los artículos 270 y siguientes del Código Penal.

La protección de esta obra se extiende al universo, de acuerdo con las leyes y convenios internacionales.

Esta obra está destinada exclusivamente para el uso particular del usuario, quedando expresamente prohibido su uso profesional en empresas, centros docentes o cualquier otro, incluyendo a sus empleados de cualquier tipo, colaboradores y/o alumnos.

Si Vd. desea autorización para el uso profesional, puede obtenerla enviando un e-mail fmarin@eidos.es o al fax (34)-91-5017824.

Si piensa o tiene alguna duda sobre la legalidad de la autorización de la obra, o que la misma ha llegado hasta Vd. vulnerando lo anterior, le agradeceremos que nos lo comunique al e-mail fmarin@eidos.es o al fax (34)-91-5017824). Esta comunicación será absolutamente confidencial.

Colabore contra el fraude. Si usted piensa que esta obra le ha sido de utilidad, pero no se han abonado los derechos correspondientes, no podremos hacer más obras como ésta.

© Marino Posadas, 2000

© Grupo EIDOS Consultoría y Documentación Informática, S.L., 2000

ISBN 84-88457-02-2

XML. Introducción al Lenguaje

Marino Posadas

Responsable editorial

Paco Marín (fmarin@eidos.es)

Coordinación de la edición

Antonio Quirós (aquiros@eidos.es)

Autoedición

Magdalena Marín (mmarin@eidos.es)

Marino Posadas (mposadas@eidos.es)

Grupo EIDOS

C/ Téllez 30 Oficina 2

28007-Madrid (España)

Tel: 91 5013234 Fax: 91 (34) 5017824

www.grupoeidos.com/www.eidos.es

www.LaLibreriaDigital.com

Índice

ÍNDICE.....	5
INTRODUCCIÓN	7
A QUIEN VA DIRIGIDA ESTA OBRA	7
REQUISITOS DEL SISTEMA.....	8
EL WORLD WIDE WEB CONSORTIUM.....	8
ORÍGENES DEL LENGUAJE XML	9
VOCABULARIOS XML	10
LOS AUTORES.....	11
¿QUÉ ES XML?.....	11
LAS DTD (DOCUMENT TYPE DEFINITIONS).....	12
UN PRIMER EJEMPLO: "HOLA MUNDO" EN XML.....	13
SOPORTE DE NAVEGADORES	14
OTRAS HERRAMIENTAS.....	14
EJERCICIOS.....	15
MECANISMOS DE AUTODESCRIPCIÓN EN DOCUMENTOS XML	17
EL CONCEPTO DE ELEMENTO EN XML	18
RESTRICCIONES SINTÁCTICAS DEL LENGUAJE XML	18
CARACTERÍSTICAS DE LAS DTD.....	19
CARDINALIDAD.....	20
EJERCICIOS.....	22
LOS DTD EN DETALLE	25

OPCIONES ADICIONALES EN LAS DECLARACIONES ELEMENT Y ATTLIST	26
LOS ESPACIOS DE NOMBRES (NAMESPACES)	29
USO DE ESPACIOS EN BLANCO	30
DTD EXTERNOS	31
EJERCICIOS.....	31
OTROS ELEMENTOS AUTODESCRIPTIVOS.....	33
RESOURCE DESCRIPTION FRAMEWORK (RDF)	33
XML-SCHEMAS	34
EJERCICIOS.....	38
XML-SCHEMAS.....	39
EJERCICIOS.....	45
MECANISMOS DE PRESENTACIÓN: HOJAS DE ESTILO (CSS) Y EL LENGUAJE DE	
HOJAS ESTILO EXTENDIDAS: XSL.....	47
¿CÓMO FUNCIONA XSL?	48
PRESENTACIÓN MEDIANTE HOJAS DE ESTILO EN CASCADA	49
PRESENTACIÓN MEDIANTE HOJAS DE ESTILO EXTENDIDAS (XSL)	51
PLANTILLAS XSL.....	52
PATRONES PARA LA DESCRIPCIÓN DE NODOS EN XSL	52
TABLA DE EQUIVALENCIA XSL PARA LOS OPERADORES RELACIONALES.....	54
EL CONJUNTO DE INSTRUCCIONES XSL.....	56
EJERCICIOS.....	58
EL MODELO DE OBJETOS DE DOCUMENTO(DOM)	59
EL MODELO DE OBJETOS DE DOCUMENTO XML	61
LOS NODOS.....	63
OBJETOS DE ALTO NIVEL DE LA JERARQUÍA DOM	63
EJERCICIOS.....	64
XML DOM EN LA PRÁCTICA	65
UN EJEMPLO COMPLETO	68
UTILIZACIÓN DE XML DOM CON HERRAMIENTAS DE DESARROLLO	70
XML DESDE VISUAL BASIC.....	70
GENERACIÓN DE FICHEROS XML A PARTIR DE RECORDSETS DE ADO	73
PROCESO DE CONSTRUCCIÓN DE UN CONVERTIDOR DE RECORDSETS DE ADO EN FICHEROS XML	74
CONCLUSIÓN.....	80
EJERCICIOS.....	80
RECURSOS XML EN INTERNET	83



1

Introducción

A quien va dirigida esta obra

Hemos preferido abordar el estudio del estándar XML desde el punto de vista de un curso (esto es, progresando paulatinamente en el nivel de complejidad) ya que dadas las características de su utilización, existen muchos aspectos que no pueden ser totalmente comprendidos si no es enfrentándose a situaciones (ejemplos, ejercicios, muestras de uso) reales.

Por eso, tras cada breve introducción teórica, le sigue un ejemplo práctico de uso, y al final de cada capítulo se proponen preguntas y ejercicios que sugiero que el lector vaya resolviendo para comprobar realmente si su grado de comprensión es suficiente para seguir adelante con aprovechamiento.

El texto pretende ser, pues, una obra de carácter práctico, que permita al lector tener un buen conocimiento de todas las tecnologías que se ven implicadas dentro de lo que genéricamente se denomina XML, de forma que se consigan unas bases sólidas sobre las que trabajar, y se tengan claros cuáles son los patrones-guía por los que debe regirse y dónde buscar cuando las cosas se tuercen.

Al tratarse de una tecnología y no de una herramienta, XML se convertirá en protagonista de escenarios muy diferentes: *Visual Basic* (la versión 7.0, anunciada para Nov/2000) se basará totalmente en ella y en el nuevo *Microsoft Forms+*, la programación Web (donde junto al nuevo *ASP+* permitirá construir soluciones Internet en las que el código de servidor sea binario –compilado– y el mecanismo de transporte de datos, XML), programación de *soluciones ofimáticas* (responde a un modelo de objetos programable utilizando el lenguaje *VBA*), utilización como *formato de intercambio* (los ficheros se escriben en texto plano y pueden ser leídos en cualquier plataforma (*Windows*, *UNIX/LINUX*, *McIntosh*, *OS/2*, *VMS*, etc.), y otros que iremos comentando.

Respecto a conocimientos previos, es importante que el lector maneje, siquiera básicamente, los fundamentos del lenguaje HTML, y para la parte final, donde se aborda el trabajo con modelos de objetos, conocimientos de algún lenguaje de programación que utilice estos modelos: Visual Basic, Delphi, Visual C++, Java, JavaScript, VBScript, Perl, etc.

Requisitos del sistema

Realmente, son muy pocos los requisitos para poder trabajar a un nivel básico con XML. Al tratarse de lenguajes de marcas, cualquier editor es válido, si bien es interesante utilizar alguno de los editores gratuitos que se encuentran en Internet, ya que efectúan validaciones del código escrito.

Por lo demás, para ver los resultados, pueden usarse esos editores, pero basta una versión actualizada de un navegador: Internet Explorer 5.x ó Netscape 6 cumplen perfectamente esa función. Solamente en la parte final, cuando se aborda la utilización desde una herramienta de desarrollo, puede ser interesante contar con alguna de las herramientas citadas anteriormente para el manejo del modelo de objetos, aunque basta tener cualquier herramienta que soporte VBA (Office, Visio, o incluso el propio Windows Scripting Host) disponible para poder hacer lo mismo, como también veremos.

El World Wide Web Consortium

Antes de nada, es preciso recordar que el organismo que establece éstos estándares es la World Wide Web Consortium (W3C), un organismo internacional de normalización, creado en 1994, similar en cierto sentido a ISO o ANSI. Sus miembros son más de 400 organizaciones de todo el mundo que corren con los gastos de financiación –aparte de algunos ingresos estatales- y lo componen profesionales de la informática de todos los sectores, cuya misión es la de definir y normalizar la utilización de los lenguajes usados en Internet, mediante un conjunto de Recommendations (recomendaciones) que son publicadas libremente, en su sitio Web (www.w3.org) y aprobadas por comités de expertos compuestos por representantes nominales de la propia W3C y técnicos especializados de las más importantes compañías productoras de software para Internet, distribuidoras y centros de investigación. Baste citar entre sus miembros más destacados a Adobe, AOL, Apple, Cisco, Compaq, IBM, Intel, Lotus, Microsoft, Motorola, Netscape, Novell, Oracle, Sun, y un largo etcétera entre las que se encuentra Iberdrola como única empresa española.

Su Web, la alberga el prestigioso **Massachusetts Institute of Technology** (M.I.T.) a través de su Laboratorio de Informática (<http://www.lcs.mit.edu/>) en EE.UU., el **INRIA** (<http://www.inria.fr/>) en Europa (Francia) y la **Universidad de Keio** en Japón (<http://www.keio.ac.jp/>). Hasta el momento, han desarrollado más de 20 especificaciones técnicas, siendo su mentor principal el propio Tim Berners-Lee (*inventor* de la WWW, y accesible en (<http://www.w3.org/People/Berners-Lee>) quien hace las funciones de jefe de equipo y trabaja en colaboración con equipos del resto de organizaciones miembro.

La primera especificación relevante publicada por la W3C fue la versión *HTML 3.2*. La última ha sido muy reciente (26/Enero/2000) y corresponde a *XHTML 1.0*, que no es sino una reformulación de HTML 4.0 en XML, que tiene como objetivos aportar una mayor flexibilidad al lenguaje permitiendo crear nuevas características y estableciendo una forma estándar de hacerlo, y por otro lado, preparar al lenguaje para el soporte de formas de acceso alternativas, llamadas también plataformas alternativas, como son la televisión, o la telefonía móvil.

Orígenes del lenguaje XML

Pero antes de continuar, debemos comentar algo sobre sus orígenes: XML se considera un subconjunto de otra especificación superior (bastante más compleja), que establece cómo deben de hacerse los lenguajes de marcas de cualquier tipo (SGML o Standard Generalized Markup Language), y que ha sido adaptada para el almacenamiento de datos. En SGML, se hace, por primera vez, distinción entre el contenido y la presentación, tal y como encontramos en la siguiente definición, sita en una página sobre SGML de la Universidad de Oviedo (<http://www6.uniovi.es/sgml.html>):

SGML permite que la estructura de un documento pueda ser definida basándose en la relación lógica de sus partes. Esta estructura puede ser validada por una Definición de Tipo Documento (DTD - Document Type Definition). La norma SGML define la sintaxis del documento y la sintaxis y semántica de DTD. Un documento SGML se marca de modo que no dice nada respecto a su representación en la pantalla o en papel. Un programa de presentación debe unir el documento con la información de estilo a fin de producir una copia impresa en la pantalla o en el papel. La sintaxis de SGML es suficiente para sus necesidades, pero pocos pueden decir que es particularmente "bella". El lenguaje muestra que se originó en sistemas donde el texto era el contenido principal y el marcado era la excepción.

Así pues, se define el estándar XML como: *El formato universal para documentos y datos estructurados en Internet*, y podemos explicar las características de su funcionamiento a través de 7 puntos importantes, tal y como la propia W3C recomienda:

1. - *XML es un estándar para escribir datos estructurados en un fichero de texto.* Por datos estructurados entendemos tipos de documentos que van desde las hojas de cálculo, o las libretas de direcciones de Internet, hasta parámetros de configuración, transacciones financieras o dibujos técnicos. Los programas que los generan, utilizan normalmente formatos binarios o de texto. XML es un conjunto de reglas, normas y convenciones para diseñar formatos de texto para tales tipos de datos, de forma que produzca ficheros fáciles de generar y de leer, que carezcan de ambigüedades y que eviten problemas comunes, como la falta de extensibilidad, carencias de soporte debido a características de internacionalización, o problemas asociados a plataformas específicas.

2. - *XML parece HTML pero no lo es.* En efecto, en XML se usan marcas y atributos, pero la diferencia estriba en que, mientras en HTML cada marca y atributo está establecido mediante un significado –incluyendo el aspecto que debe tener al verse en un navegador–, en XML sólo se usan las marcas para delimitar fragmentos de datos, dejando la interpretación de éstos a la aplicación que los lee.

3. - *XML está en formato texto, pero no para ser leído.* Esto le da innumerables ventajas de portabilidad, depuración, independencia de plataforma, e incluso de edición, pero su sintaxis es más estricta que la de HTML: una marca olvidada o un valor de atributo sin comillas convierten el documento en inutilizable. No hay *permisividad* en la construcción de documentos, ya que esa es la única forma de protegerse contra problemas más graves.

4. - *XML consta de una familia de tecnologías.* Por supuesto, existe una definición (estándar) de XML 1.0 que viene de Febrero 98, pero su desarrollo se ha ido enriqueciendo paulatinamente a medida que se veían sus posibilidades: de esa forma, contamos con una especificación Xlink, que describe un modo estándar de añadir hipervínculos a un documento XML. XPointer y Xragments son especificaciones para establecer la forma de vincular partes de un documento XML. Incluso el lenguaje de hojas de estilo (CSS) se puede utilizar con XML al igual que se hace con HTML. XSL es precisamente, una extensión del anterior, en la que se dispone de todo un lenguaje de programación exclusivamente para definir criterios de selección de los datos almacenados en un documento XML, y que funciona conjuntamente con las CSS o con HTML para suministrar al programador y al usuario mecanismos de presentación y selección de información, que no requieran de la intervención constante

del servidor. Se basa en un lenguaje anterior para transformación (XSLT) que permite modificar atributos y marcas de forma dinámica.

El Modelo de Objetos de Documento (DOM) es un conjunto estándar de funciones para manipular documentos XML (y HTML) mediante un lenguaje de programación. XML Namespaces, es una especificación que describe cómo puede asociarse una URL a cada etiqueta de un documento XML, otorgándoles un significado adicional. Y finalmente, XML-Schemas es un modo estándar de definir los datos incluidos en un documento de forma más similar a la utilizada por los programadores de bases de datos, mediante los metadatos asociados. Y hay otros en desarrollo, pero todos están basados en el principal: XML.

5. - *XML es prolijo, pero eso no supone un problema.* Los ficheros resultantes, son casi siempre mayores que sus equivalentes binarios. Esto es intencionado, y las ventajas ya las hemos comentado más arriba, mientras que las desventajas, siempre pueden ser soslayadas mediante técnicas de programación. Dado el reducido coste actual del espacio en disco y la existencia gratuita de utilidades de compresión, junto al hecho de que los protocolos de comunicación soportan sistemas rápidos de compresión, este aspecto no debe resultar problemático.

6. - *XML es nuevo, pero no tanto.* El estándar empezó a diseñarse en 1996, y se publicó la recomendación en Febrero/98. Como ya hemos comentado, eso no significa que la tecnología no esté suficientemente madura, ya que el estándar SGML en el que se basa, data de una especificación ISO del año 1986.

7. - *XML no requiere licencias, es independiente de la plataforma, y tiene un amplio soporte.* La selección de XML como soporte de aplicaciones, significa entrar en una comunidad muy amplia de herramientas y desarrolladores, y en cierto modo, se parece a la elección de SQL respecto a las bases de datos. Todavía hay que utilizar herramientas de desarrollo, pero la tranquilidad del uso del estándar y de su formato, hacen que las ventajas a la larga sean notables.

Tenemos pues, dos partes bien definidas dentro de todo documento XML: la *definición* de contenidos y los propios contenidos (el DTD y los datos). Cada definición, o DTD constituye de por sí una forma de escribir documentos para Internet.

Vocabularios XML

Según esto, e independientemente de que exista una familia entera de tecnologías asociada a XML, cada rama del conocimiento humano es susceptible de establecer una normativa de escritura que establezca con precisión como deberían estar escritos los documentos relativos a ella, que recoja con precisión cómo interpretar sus peculiaridades sin imprecisiones y confusiones semánticas.

Así nos encontramos con estándares (un maremagno de siglas, la verdad), que tienen que ver con Informática, como CDF (Channel Definition Format) para la definición de canales en servidores push, o VRML (Lenguaje de marcas para Realidad Virtual), pero también podemos hallar definiciones para las actividades más variadas, tales como MathML (para la descripción de datos de tipo matemático), CML ó Chemical Markup Language (para la descripción de datos sobre Química), GenXML (para datos genealógicos), y un larguísimo etcétera, entre las que figuran especificaciones tan curiosas como OML (Lenguaje de marcas para Ontología), VoxML (para la descripción de la voz humana), LitML (Lenguaje de marcas para Liturgia), BRML (para reglas de negocio), HRMML (para la gestión de Recursos Humanos), SML (Lenguaje de marcas para las industrias del acero) o MRML (para documentos sobre la lectura del pensamiento).

En la dirección de Internet <http://www.oasis-open.org/cover/siteIndex.html>, puede consultarse la lista (larguísima), de todos estos estándares albergada en el sitio web de otra de las entidades que están trabajando activamente por la estandarización de los recursos de la Web: OASIS¹

Los autores

Aunque hemos hablado de W3C y su equipo de trabajo, aparte del conocidísimo **Tim Berners-Lee** no hemos citado a nadie más. Quizá para algún lector sería interesante conocer qué influencias ha recibido el estándar en su creación debido al origen de los firmantes de la recomendación. Bien, pues dado al carácter abierto de los grupos de trabajo, las influencias son muy variadas: Tim Bray (*Netscape*) y Jean Paoli (Microsoft) figuran junto a C.M. Sperberg-McQueen (Universidad de Illinois) como editores, pero el grupo de trabajo incluía a: John Bosak (Sun Microsystems), Dan Conolly (W3C), Dave Hollander (Hewlett-Packard), Tom Magliery (NCSA), M. Murata (Fuji Xerox Information Systems) y Joel Nava (Adobe) entre otros.

Todos ellos son ahora bien conocidos dentro del mundillo XML y casi todos disponen de sus propias páginas Web explicando aspectos adicionales del lenguaje. Tim Bray, por ejemplo, ha publicado una edición comentada del estándar (muy didáctica), que pueden encontrarse en la dirección: <http://www.xml.com/axml/testaxml.htm>.

Y lo mismo puede decirse de los otros estándares asociados al lenguaje XML que hemos citado anteriormente, y cuya lista sería prolija de mencionar aquí. Baste decir que las listas de los grupos de trabajo (más largas para cada recomendación publicada), contienen a representantes de casi todos las compañías y con ello, se adquiere un valor de consenso muy alto.

¿Qué es XML?

Si observamos con atención, todo indica que los avances que se producen en las tecnologías de la información parecen basarse siempre en una misma forma de proceder: sacar factor común. Los sistemas operativos gráficos hacen eso cuando separan un conjunto de funciones en ficheros DLL que son compartidos por la mayoría de las aplicaciones; las API de ODBC y OLE-DB son una forma común de acceder a datos cuyos orígenes y formatos son bien diversos, pero que son vistos por el programador a través de una misma jerarquía de objetos (DAO, RDO ó ADO), independientemente del sistema de soporte; Las máquinas virtuales de Java no pretenden otra cosa que permitir que código escrito en un mismo lenguaje pueda ser interpretado en diferentes plataformas, cuya estructura es bien distinta; VBA y los modelos de objetos posibilitan la programación de macros en herramientas tan diversas como Autocad 14 ó Microsoft Office mediante el mismo lenguaje. Incluso ActiveX y la tecnología en 3 capas, no hace sino extraer aquella parte de los programas que puede residir en una máquina distinta y -dotándole de una estructura estándar- permitir que se exponga como objeto programable por otras herramientas.

Por ello no es de extrañar que los principios que han animado la creación del estándar XML sean los mismos: separar dos partes que hasta ahora estaban indisolublemente unidas en un documento HTML: la presentación y los datos que son presentados. Es cierto que existe un paso previo en este proceso de desligamiento: las hojas de estilo en cascada ó CSS (Cascade Style Sheets); pero éstas solamente

¹ Al final de esta obra podrá encontrar el lector un apéndice dedicado a algunos de los recursos sobre XML que se encuentran en Internet

permiten definir patrones de presentación independientes que son aplicados mediante etiquetas HTML a un documento. Se trata de una separación conceptual, que no factual: los datos siguen estando en la página HTML.

Con XML la separación es total. Podemos crear un conjunto de hojas de presentación (ficheros XSL, un derivado del estándar) y aplicarles en diferentes contextos el mismo documento XML que contiene los datos: obtendremos tantas presentaciones de éstos como ficheros XSL utilicemos. Además, el estándar XSL es más potente que las hojas de estilo en cascada, pues permite delimitar qué conjuntos de información deseamos y hasta el orden en que queremos que aparezcan.

No es, pues, otro lenguaje de marcas para Internet. XML es un meta-lenguaje de marcas, es decir, permite que el usuario diseñe sus propias marcas (tags) y les dé el significado que se le antoje, con tal de que siga un modelo coherente. La primera gran ventaja de XML es el hecho de que los datos se auto-definen a sí mismos y esa definición pueden encontrarse en la propia página XML (ó en otra separada a la que se hace referencia), suministrando así a cualquier programa que abra ese fichero la información necesaria para el manejo de su contenido. De hecho, un fichero XML correctamente escrito requiere dos cualidades que establecen su grado de conformidad con las reglas establecidas:

Fichero **XML bien formado** (well formed) es aquel que se ha escrito de acuerdo con el estándar.

Fichero **XML válido** es aquel que -cumpliendo con la definición del estándar- está, además, lógicamente bien estructurado y define en su totalidad cada uno de sus contenidos sin ambigüedad alguna.

Por si esto fuera poco, su formato (texto plano) permite su transporte y lectura bajo cualquier plataforma o herramienta y le da un valor de universalidad que no se puede conseguir con ningún formato nativo, por mucha aceptación que tenga. Incluso su convivencia con el código HTML actual está garantizada gracias a que existe una marca HTML definida con la sintaxis `<XML>Fichero XML</XML>`, que permite definir una ubicación de un fichero XML externo que puede ser embebido en el documento y tratado de forma conjunta. Durante largo tiempo (...), se prevé una coexistencia de ambos, pero bien entendido, que HTML sólo es un caso particular de XML, donde su DTD define cómo representar los documentos en un navegador.

Recientemente, se ha definido un nuevo estándar por parte de la W3C que recibe el nombre de XHTML. Se trata, utilizando su propia definición, de “una reformulación del lenguaje HTML usando la sintaxis XML”. Esto supone que aquellos documentos escritos en XHTML deberán de guardar las restricciones del lenguaje que impone XML, aunque el lenguaje en sí, siga siendo HTML. Esto quiere decir que no se permiten etiquetas sin cerrar, atributos sin entrecomillar, etc. Su propósito es normalizar más fuertemente el lenguaje HTML de cara a las nuevas generaciones de navegadores, permitiendo la integración de los dos lenguajes y evitando las ambigüedades existentes en la actualidad.

Las DTD (Document Type Definitions)

Si queremos tener la seguridad de que un fichero XML pueda ser perfectamente interpretado por cualquier herramienta, podemos (aunque no es obligatorio) incluir una definición de su construcción que preceda a los datos propiamente dichos. Este conjunto de meta-datos recibe el nombre de DTD ó *Definición de Tipo de Documento*. Esta definición sí que debe basarse en una normativa rígida, que es la definida por XML.

Así, si una herramienta es capaz de interpretar XML, eso significa que posee un analizador sintáctico (*Parser*) que es capaz de contrastar la definición dada por el autor del documento contra la especificada por la normativa, indicando si hay errores, y, de no ser así, presentando el documento de

la forma adecuada. Esto es por ejemplo lo que hace Internet Explorer 5.0, o Netscape 6.0, cuando abren un documento XML.

Por lo tanto, las DTD's son la clave de la auto-descripción: siguiendo el modelo establecido por XML, permiten definir al usuario qué significa exactamente cada una de las marcas que va a incluir a continuación para identificar los datos. De ella, se derivan otras ventajas, como la posibilidad de soslayar el modelo simétrico de datos (el formato *clásico* de datos tiene que tener forma rectangular: filas y columnas), permitiendo diseñar modelos de datos totalmente jerárquicos que adopten la forma de árboles asimétricos si ello describe el contenido de forma más precisa, y evitando repeticiones.

La estructura de árbol se presenta, pues, más rica en posibilidades de representación que la tabular, permitiendo representar la información de forma más exacta.

Un primer ejemplo: "Hola mundo" en XML

Siguiendo la costumbre que inició Charles Petzold en su *Programación en Windows* comencemos por lo más simple: veamos un primer ejemplo, en el Código Fuente1, del mínimo programa que muestra en pantalla el mensaje "Hola Mundo".

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Mensaje [ <!ELEMENT Contenido (#PCDATA)> ]>
<!-- este es un comentario -->
<Contenido>¡Hola, mundo!</Contenido>
```

Código Fuente 1. "Hola Mundo" en XML.

En el ejemplo ya podemos observar 3 líneas clave: La primera, es la definición general. Nos indica que lo que viene a continuación es un documento XML (las de inicio y fin son el carácter obligatorio que delimita esa definición. Además, observamos dos atributos: *versión* -que se establece a 1.0- que nos indica que el intérprete de XML debe de utilizar las normas establecidas en Febrero/98 y *encoding*, asignado a "UTF-8", y que el estándar recomienda incluir siempre, aunque algunos navegadores (como Explorer 5) no lo exijan de forma explícita.

Téngase en cuenta que XML debe soportar características internacionales, por tanto se dice que, tras su interpretación, todo documento XML devuelve Unicode. El valor por defecto es "UTF-8".

La segunda línea es una DTD muy simple. Consta de la declaración de tipo de documento mediante !DOCTYPE seguido del nombre genérico que va a recibir el objeto que se defina a continuación (mensaje), e indica que sólo va a contener un elemento (!ELEMENT) que también se denominará mensaje y que está compuesto de texto (#PCDATA).

Finalmente, la cuarta línea (la tercera es un simple comentario) contiene la información en sí. Dentro de dos etiquetas de apertura y cierre con el nombre definido en la línea 2, se incluye la información propiamente dicha. Si visualizamos el documento en Internet Explorer 5.0 obtendremos una salida como la de la Figura 1.

Piense el lector que en ésta salida no estamos indicando ningún modo de presentación. Por tanto IE5 asume que lo que queremos es analizar el documento con el *parser* y averiguar si existe algún error en él: reconoce el tipo de documento, simplifica el DTD limitándose a mostrar su cabecera, y recorre los datos cambiando el color de las marcas y símbolos para que la interpretación sea más sencilla.

Eventualmente, será capaz de mostrar cualquier jerarquía en formato auto-desplegable (Estilo Tree-View).

A partir del mes próximo profundizaremos más en la construcción de ficheros XML, en su presentación mediante ficheros de formato XSL, (ó CSS) y en su manipulación desde Explorer, para continuar con el manejo a través de la librería MSXML desde Visual Basic 6.0, y la forma de integrarlo en la tecnología 3 capas.

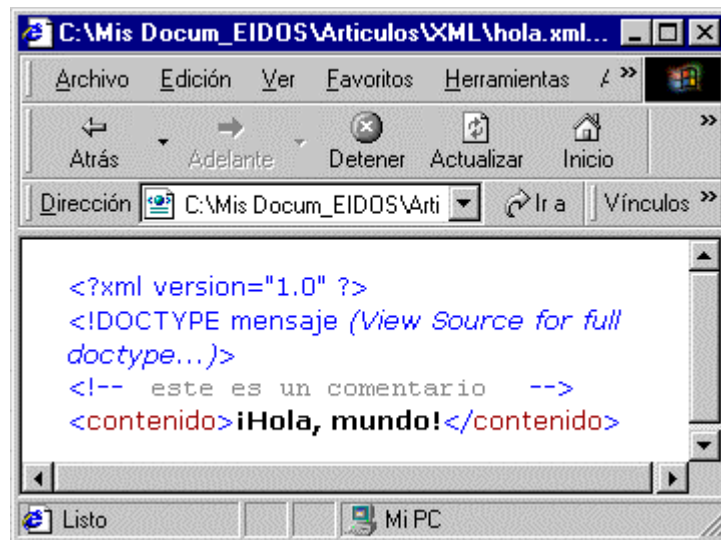


Figura 1. Salida del código XML anterior interpretado por Internet Explorer 5.0

Soporte de Navegadores

Las versiones de Internet Explorer 5.x, Mozilla 5.0 (la versión de Netscape de código abierto) y Netscape 6.0 ofrecen diferentes grados de soporte del estándar XML. En el caso de Explorer, el soporte es bastante completo² y permite -además- abrir documentos de formato XSL que evalúa e interpreta sintácticamente para garantizar que no hay errores de composición e indicando esta circunstancia si se produce (incluye *parser* y depurador), lo que le convierte en una herramienta de trabajo en sí misma para el desarrollo con XML. Lo mismo sucede con Mozilla 5.0 y con Netscape 6.0 si bien posteriores análisis determinarán el grado de conformidad con el estándar.

Otras herramientas

Desde el año 98, están apareciendo herramientas para ayudar al desarrollador en la edición, análisis y depuración de documentos XML, de las cuales, una buena parte son gratuitas ó de carácter *shareware*. Entre las que hemos probado, destacan **XML-Spy**, **XML-Writer**, y la suite **Visual Tools** de **IBM**, que es un conjunto de herramientas para el trabajo con XML, que puede descargarse de su sitio web *AlphaWorks*. En la lista de recursos de Internet, puede el lector encontrar varias direcciones que contienen dicho software, así como páginas de enlaces con otros que continuamente van apareciendo.

² Para un análisis más detallado, véase el artículo de David Brownell sobre el analizador MSXML incluido con Explorer 5.0, en la dirección: <http://www.xml.com/pub/1999/11/parser/index.html>

Ejercicios

1. ¿Es el XML una versión de HTML, o un lenguaje independiente?
2. ¿En qué formato se escriben los documentos XML?
3. ¿Qué ventajas aporta respecto al lenguaje de marcas tradicional?
4. ¿Es un sustituto de XML, una extensión, o un complemento?
5. ¿Que mecanismos existen para interpretar un documento XML?
6. ¿Los navegadores soportan ese lenguaje actualmente?
7. ¿Se puede utilizar XML fuera de entornos web?
8. ¿Qué empresa o entidad es la responsable de la creación de XML?
9. ¿Qué es básicamente un DTD?
10. ¿En qué consisten sucintamente los vocabularios XML?
11. ¿Se usan algunos de esos vocabularios en la actualidad?

Mecanismos de autodescripción en documentos XML

En el capítulo anterior, veíamos cómo definir un documento muy simple (una versión del "Hola Mundo"), mediante un DTD de una sola línea y cuál era su apariencia posterior al mostrarse en el navegador Internet Explorer 5.0. Para definir ese DTD utilizábamos la instrucción que aparece en el Código Fuente 2.

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

Código Fuente 2. Código del DTD inicial para "Hola Mundo"

Podemos concebir el papel de los DTDs de modo similar al de los llamados *metadatos* en otras aplicaciones como SQL-Server u Oracle. Los metadatos son información acerca de la estructura de los propios datos, que nos ayudan a entender su arquitectura. Son, por tanto, una descripción de la organización que se ha usado al escribir los datos en sí.

No obstante, los DTDs resultan complejos en su sintaxis y no muy evidentes en su descripción, por lo que existe otro estándar de descripción en fase de estudio que Internet Explorer 5.0 ya implementa, denominado *XML-Data* ó *XML-Schemas*. La diferencia de éste último con los DTDs, aparte de resultar mucho más descriptiva que aquellos, es que XML-Schemas se expresa con una sintaxis pura XML.

Y es que, como puede apreciar el lector, existen diferencias de estructura sintáctica importantes entre el clásico HTML y XML. Además, una de ellas estriba en que para el lenguaje XML, ciertas *concesiones* que estaban permitidas en HTML ya no son válidas.

Vamos a hacer un recuento de las diferencias fundamentales, pero antes, un recordatorio previo: la terminología que utilizamos en la descripción de las partes de un documento (tanto HTML como XML) que podemos resumir en la Figura 2.

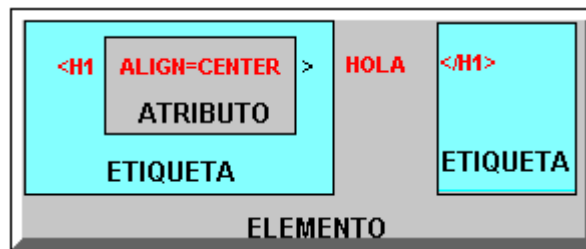


Figura 2. Partes de un documento

El concepto de elemento en XML

Por elemento entendemos el conjunto de fragmentos que componen la etiqueta o *tag* HTML: La etiqueta de cabecera o identificador, los atributos, el cuerpo o contenido y la etiqueta de cierre.

Restricciones sintácticas del lenguaje XML

1. No se permite la anidación incorrecta de elementos (Deben cerrarse correctamente, según el grado de profundidad de la anidación).
2. No se permiten elementos sin etiqueta de cierre (etiquetas cuyo cierre era opcional en HTML)
3. Los elementos que no poseen contenido (etiquetas del tipo `<HR SIZE="3">`), deben de utilizar una etiqueta de cierre, o usar la abreviatura permitida en XML, consistente en incluir una barra vertical (/) antes del carácter de cierre (>). Por ejemplo, en el caso anterior tendríamos que expresarlo mediante `<HR SIZE="3" />`.
4. Todos los atributos deben de ir entre comillas dobles (como en el ejemplo anterior).
5. XML diferencia entre mayúsculas y minúsculas (quizás se trata de la característica más incómoda al principio, pues produce errores en los analizadores sintácticos, que pueden ser difíciles de detectar).

Por otro lado, la implementación del parser HTML de Internet Explorer 4.01/5.0 soporta la inclusión de una etiqueta `<XML>` (con su correspondiente etiqueta de cierre), en la cual podemos incluir documentos enteros escritos en XML y que serán tratados adecuadamente de forma separada del Código HTML, si bien lo más normal es que -si incluimos una etiqueta XML en un documento HTML- lo hagamos para referenciar un documento externo XML que será cargado en memoria y procesado, lo que es preferible por razones de claridad.

A ésta técnica se le denomina inclusión de islas de datos (*data islands*), y podemos expresarla como indica el Código Fuente 3.

```
<XML ID="Origen_de_datos" SRC="Datos.XML"></XML>
```

Código Fuente 3. Inserción de una Data-island de XML en un documento HTML

El comportamiento del analizador será el siguiente: cuando encuentra una etiqueta `<XML>` en un documento, asume que el contenido será un conjunto de datos con una estructura, y crea un recordset de solo lectura en la memoria a partir de la información leída del fichero externo *Datos.xml*, que debe residir en un directorio accesible por la página Web que se está interpretando.

A su vez, éste recordset, de nombre *Origen_de_datos*, puede programarse mediante cualquiera de los lenguajes de script soportados por el navegador: JavaScript o VBScript.

Características de las DTD

Pero volvamos a la descripción más detallada de la construcción de las DTD. El estándar define un conjunto limitado de identificadores para la creación de estructuras de datos: DOCTYPE, ELEMENT, y ATTLIST.

Como todo documento XML tiene que incluir un nodo raíz (recuerde el lector que XML extiende la metáfora de estructura tabular -filas y columnas-, a una estructura jerárquica), DOCTYPE define el nombre del nodo raíz del árbol jerárquico, así como el punto de inicio del documento XML. También sirve para la declaración del origen del propio DTD.

El identificador ELEMENT sirve para declarar cada uno de los elementos de un DTD. Sirve tanto para la descripción de elementos simples como de elementos compuestos (de otros elementos). Dispone de un grupo de símbolos asociados que permiten al diseñador indicar la cardinalidad de los elementos (si puede existir uno ó más de uno, uno o ninguno, etc.), así como de especificaciones consistentes en palabras reservadas, que informan acerca del tipo de dato, si es requerido ó no, si tiene un valor por defecto, si puede adoptar cualquier otra forma, etc. Finalmente, el identificador ATTLIST permite la definición de listas de atributos para los elementos, pudiendo también incluir datos sobre los valores aceptables para ese elemento, y valores por defecto. En el Código Fuente 4 vemos un ejemplo de implementación a partir de la base de datos Pedidos2000 utilizada como origen de registros.

```
<?xml version="1.0" encoding="UTF-8"?>
<Clientes>
<Cliente>
  <IdCliente>1</IdCliente>
  <Empresa>Defensa Personal Orient</Empresa>
  <Contacto>Alfonso Papo</Contacto>
  <Ciudad>Terrassa</Ciudad>
  <Provincia>Barcelona</Provincia>
</Cliente>
<Cliente>
  <IdCliente>2</IdCliente>
  <Empresa>Mística y Salud S.A.</Empresa>
  <Contacto>Ana Coreta</Contacto>
  <Ciudad>Santander</Ciudad>
  <Provincia>Santander</Provincia>
</Cliente>
<Cliente>
  <IdCliente>3</IdCliente>
  <Empresa>Paños Rias Bajas</Empresa>
```

```

    <Contacto>Estela Marinera</Contacto>
    <Ciudad>Vigo</Ciudad>
    <Provincia>Pontevedra</Provincia>
</Cliente>
</Clientes>

```

Código Fuente 4. Fichero de datos XML "plano" (sin DTD)

En este mismo Código Fuente 4 podemos ver que en el fichero adjunto, el nodo raíz se denomina `<Clientes>` y está compuesto de una serie de elementos `<Cliente>`, que son elementos compuestos, a su vez, de cinco elementos simples: `<IdCliente>`, `<Empresa>`, `<Contacto>`, `<Ciudad>` y `<Provincia>`.

Según esa estructura de datos, el DTD que le correspondería a este fichero tendría la definición que aparece en el Código Fuente5

```

<!DOCTYPE Clientes [
<!ELEMENT Clientes (Cliente+)>
<!ELEMENT Cliente (IdCliente, Empresa, Contacto, Ciudad, Provincia)>
<!ELEMENT IdCliente (#PCDATA)>
<!ELEMENT Empresa (#PCDATA)>
<!ELEMENT Contacto (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
<!ELEMENT Provincia (#PCDATA)>
]>

```

Código Fuente 5. DTD correspondiente al fichero XML anterior

De este fichero podemos inferir la lógica de su construcción. DOCTYPE, como se decía, define el propio punto de inicio del DTD: tanto su nombre como el del elemento raíz. Una vez definido, se desglosa la lista de elementos, comenzando por el raíz (**Clientes**), que está compuesto de elementos **Cliente** (el signo + significa que el elemento puede aparecer una ó más veces). A su vez, cada elemento Cliente, se compone de cinco elementos: **IdCliente**, **Empresa**, **Contacto**, **Ciudad** y **Provincia**.

Al lado de la definición de cada elemento individual, aparece la definición del tipo de dato que puede contener ese elemento: `#PCDATA` significa *Parsed Character Data*, e indica que el valor contenido entre las etiquetas serán caracteres estándar (esto es pasados por el analizador sintáctico, que eliminará todos aquellos prohibidos por la especificación: `<`, `>`, `,`, etc. También podríamos haber usado otros atributos de definición de datos, como los de datos binarios.

Cardinalidad

En la Tabla 1 aparecen los símbolos de cardinalidad en la definición de estructuras.

Símbolo	Significado
+	el elemento puede aparecer una o más veces
*	el elemento puede aparecer cero o más veces

?	que el elemento puede aparecer cero o una vez
	el elemento puede aparecer una vez a escoger de una lista enumerada

Tabla 1. Símbolos de Cardinalidad

En el Código Fuente 5, Cliente puede aparecer por tanto, una o más veces, mientras que en el DTD alternativo del Código Fuente 6, Clientes está compuesto de elementos Cliente, los cuales pueden no aparecer o hacerlo más de una vez.

```
<!ELEMENT Clientes (Cliente*)>
<!ELEMENT Cliente (Nombre, Apellidos)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Apellidos (#PCDATA)>
```

Código Fuente 6. Fragmento de un DTD con elementos de cardinalidad y elementos compuestos

A su vez, cada elemento Cliente se compone de dos elementos anidados: Nombre y Apellidos.

También podemos ofrecer una alternativa controlada a la representación: por ejemplo si lo que queremos es extender las posibilidades de expresión del elemento Clientes de modo que pueda servir para una lista de Clientes, o para una referencia simple, o incluso para un cliente desconocido, podríamos modificar el fragmento del DTD asociado utilizando lo que se conoce como una lista enumerada. Dicha lista es una secuencia de definiciones separadas por el signo |, de las cuales el elemento tendrá que asumir obligatoriamente uno de ellos.

De esa forma la declaración aparecería como se muestra en el Código Fuente 7

```
<!ELEMENT Clientes (Cliente* | Referencia | Anonimo)>
<!ELEMENT Cliente (Nombre, Apellidos)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Apellidos (#PCDATA)>
<!ELEMENT Referencia (#PCDATA)>
<!ELEMENT Anonimo EMPTY>
```

Código Fuente 7. Modificación del Fragmento de DTD anterior para soporte de una lista enumerada

La clave **EMPTY** significa que el elemento no tiene contenido, y la mayor parte de las veces se utiliza en elementos en los cuales los datos se han preferido definir como parte de un atributo.

En el Código Fuente 8 y 9 observamos como podemos incluir en los datos valores válidos de Clientes.

```
<Clientes>
<Referencia>Cliente en tratos directos con la Gerencia<Referencia>
</Clientes>
```

Código Fuente 8. Fragmento de datos válidos correspondiente a la nueva definición <Clientes>

```
<Clientes>
<Anonimo />
</Clientes>
```

Código Fuente 9. Fragmento de datos válidos correspondiente a la nueva definición <Clientes>

Otra posibilidad es declarar el elemento como ANY, que indica que el elemento puede aceptar cualquier tipo de contenido. Según esto, cualquiera de los contenidos de datos que aparecen en el Código Fuente 10 son válidos.

```
<Anonimo />
<Anonimo>ANÓNIMO</Anonimo>
<Anonimo>
  <DETALLES>ANÓNIMO</DETALLES>
</Anonimo>
```

Código Fuente 10. Alternativas de Datos válidos correspondientes a la definición <!ELEMENT Anonimo ANY>

Notará el lector que en este último caso, incluso es posible añadir una etiqueta *ad-hoc*, esta es creada a propósito para ese elemento, y que no tiene que repetirse necesariamente, ya que para otra aparición de <Anonimo> podíamos decantarnos por otra etiqueta diferente. Esta declaración ofrece el mayor grado de libertad, aunque, como es lógico, también el mayor grado de ambigüedad y de posibles problemas, de cara a la aplicación de deba, posteriormente, leer los datos e interpretarlos.

Ejercicios

1. ¿Son los DTD similares a los metadatos? ¿En qué sentido?
2. ¿Es imprescindible la presencia de un DTD en un documento?
3. ¿Cuál es el comportamiento de un navegador al leer un documento de extensión XML sin la presencia de un DTD?
4. ¿En qué lenguaje se escriben los DTD?
5. Construye un DTD para un documento XML que contenga la siguiente estructura de datos:

```
<CASA PORTAL="1-A">
<COCINA>25m.</COCINA>
<ASEO1>10m</ASEO1>
<ASEO2>15m</ASEO2>
<SALON>35m</SALON>
<DORMITORIO1>18m<DORMITORIO1>
<DORMITORIO2>24m<DORMITORIO2>
</CASA>
<CASA PORTAL="1-B">
<COCINA>25m.</COCINA>
<ASEO1>10m</ASEO1>
<ASEO2>15m</ASEO2>
<SALON>35m</SALON>
<DORMITORIO1>18m<DORMITORIO1>
```

```
<DORMITORIO2>24m<DORMITORIO2>  
</CASA>
```

6. ¿Es válida la siguiente declaración XML? : <Dato>Contenido<dato>
7. ¿Qué símbolo se utiliza para indicar que un elemento debe aparecer 0 ó 1 vez?
8. ¿Qué declaración incluiríamos en un DTD para indicar que el elemento <GARAGE /> está vacío?
9. Exactamente, ¿qué define DOCTYPE?

Los DTD en detalle

Vamos a completar nuestra definición original analizando la tercera declaración para la construcción de DTD: **ATTLIST**. Esta palabra reservada permite añadir atributos a una etiqueta definida por nosotros mediante una declaración **ELEMENT**. En el Código Fuente 11 aparece una típica definición de atributos, donde la secuencia **{elemento} {nombre-atributo} {tipo-de-dato} {tipo-de-atributo} 'valor-por-defecto'**, puede repetirse todas las veces que sea necesario para crear atributos adicionales para un elemento dado, siendo obligatoria la inclusión, al menos, del **{nombre-atributo}** y **{tipo-de-dato}**. El **{tipo-de-atributo}** y el **'valor-por-defecto'** sólo se incluyen cuando es necesario.

```
<!ATTLIST {elemento} {nombre-atributo} {tipo-de-dato} {tipo-de-atributo} 'valor-por-defecto' >
```

Código Fuente 11. Declaración formal de un atributo

Como ejemplo podemos ver, en el Código Fuente 12, lo que sería la definición de los atributos del elemento **IMG** de **HTML**, que permiten definir el origen del gráfico, y su alineación.

```
<!ATTLIST IMG SRC CDATA #IMPLIED ALIGN (left | right | center) "left" >
```

Código Fuente12. Declaración de un atributo para el elemento **IMG**

Veamos el significado de la declaración anterior: Definimos dos atributos sobre la etiqueta **IMG**: el primero se llama **SRC**, esta formado por caracteres y puede contener cualquier cadena. El segundo, se

llama `ALIGN` y su valor sólo puede ser uno a escoger de la lista enumerada que se expresa a continuación, tomándose por defecto el valor *"left"* en caso de no indicarse ninguno (se admite el entrecomillado simple o doble). En principio, no existe límite al número de atributos definibles mediante esta sintaxis.

Opciones adicionales en las declaraciones `ELEMENT` Y `ATTLIST`

Como hemos visto en la declaración anterior, `SRC` iba acompañado de dos calificadores: `CDATA` e `#IMPLIED`. El primero, corresponde a la definición del {tipo-de-dato}, la segunda al {tipo-de-atributo}. Los tipos de datos posibles según el estándar los vemos en la Tabla 2.

Tipo-de-dato	Identificador
Texto plano	CDATA
Identificador único	ID
Entidad no-textual	ENTITY
Predefinido	(Valor Valor [...] Valor)

Tabla 2. Tipos de datos con sus identificadores

`CDATA` significa que lo que sigue son caracteres en texto plano, sin analizar, y por lo tanto pueden contener caracteres reservados de XML que no serían válidos en las declaraciones `#PCDATA` que veíamos en anteriores DTD (como los símbolos `<`, `>`, `"`, `'`). En este caso, eso es necesario, ya que el valor del atributo consistirá en una URL encerrada entre comillas, que indicará la ubicación del fichero gráfico. Otra opción es utilizar las denominadas entidades de referencia a caracteres (`>`, `&`, `'`, etc.) usadas en HTML para permitir la inclusión de caracteres especiales. Pero hay casos en los que puede ser más clara esta posibilidad.

Por ejemplo, imaginemos que un fichero XML contiene un dato que especifica una sentencia SQL, que deberá usarse para una consulta. Si la sentencia tiene la forma `SELECT * FROM CLIENTES WHERE CIUDAD='MADRID' AND SALDO < 0`, el analizador devolvería errores al encontrarse con las comillas simples y con el símbolo `<`. Una forma de solucionarlo sería, entonces, expresarlo como aparece en el Código Fuente 13.

La secuencia de caracteres `![CDATA[` forzará al compilador a interpretar toda la línea como caracteres que no debe analizar, excluyendo la terminación `]]>` y dejando la sentencia `SELECT` que se encuentra entre los corchetes, intacta.

```
<Sentencia SQL>
  <![CDATA[SELECT * FROM CLIENTES WHERE CIUDAD='MADRID' AND SALDO < 0]]>
</Sentencia SQL>
```

Código Fuente 13. Datos del tipo `CDATA`

ID (*Identification*) es un nombre único (no podrá repetirse para ningún otro elemento del documento), que permitirá al *parser* convertir el elemento en un objeto programable mediante hojas de estilo o lenguajes de *script*. Se trata de una característica que ya habíamos visto en HTML dinámico: crear objetos en memoria a partir de elementos HTML, para permitir su manipulación en servidor o en cliente mediante un lenguaje de *script*.

ENTITY es, -de forma simplificada- un alias que asocia un nombre único a un conjunto de datos. Los datos asociados con el nombre pueden, sin embargo, tomar una de las formas que aparecen en la Tabla 3.

Tipo de Entidad	Funcionamiento
Interna	Hace corresponder un nombre con una frase definida en la declaración de la entidad
Externa textual	Igual que el anterior, pero la frase reside en un fichero externo
Externa binaria	Hace corresponder un nombre con un fichero externo de tipo binario o no-XML, tal como un gráfico, un documento con formato, un fichero multimedia o un applet de Java.
Ref. Caracteres o números	Usados para incluir caracteres no-ASCII en documentos XML

Tabla 3. Tipos de Entidades con su funcionamiento

En el primer caso, podemos tener la declaración de una entidad empresarial, y usarla después en el resto del documento utilizando su alias. Es el caso del Código Fuente 14.

```
<!ENTITY Eidos "Grupo EIDOS S.L." >
```

Código Fuente 14. Declaración de una entidad textual básica

y luego, como hacemos en el Código Fuente 15, utilizarla en la parte de datos del documento.

```
<Empresa>&Eidos;</Empresa >
```

Código Fuente15. Uso de una entidad textual básica

Y el analizador sustituirá el contenido &Eidos; por el definido en la entidad, *Grupo EIDOS S.L.* En el caso de una entidad externa, la referencia se hace a una ubicación y un fichero concretos. En el Código Fuente 16 tenemos un ejemplo de como incluir el logotipo de la empresa.

```
<!ENTITY LogoEidos SYSTEM "Graficos/LogoEIDOS.gif" NDATA GIF >
```

Código Fuente 16. Declaración de una entidad binaria externa

En este caso, la declaración de la entidad, LogoEidos, va seguida de SYSTEM, y la URI del fichero correspondiente, más la palabra reservada **NDATA** y la notación de tipo de dato GIF. SYSTEM indica que el fichero es externo, NDATA, que el fichero es binario, y GIF es una declaración **NOTATION**, utilizada en los casos en que se desea asociar a una extensión de fichero, un programa que lo maneje, en caso de que el procesador de fichero XML no supiese qué hacer con él.

La declaración anterior ya está asociada a un programa, pero si quisiéramos declarar una asociación alternativa, utilizando el viejo programa PAINT del sistema, podríamos hacerlo mediante la declaración que aparece en el Código Fuente 17.

```
<!NOTATION GIF SYSTEM "PAINT.EXE" >
```

Código Fuente 17. Declaración de una notación (NOTATION)

Finalmente, la cuarta opción -que no lleva delimitador- es la de la lista enumerada, en la que obligamos a que el valor expresado sea uno de los que aparecen en la lista. Hay que notar que en este caso existen dos posibilidades: si no añadimos un valor por defecto el atributo puede no suministrarse, pero si lo hacemos aparecerá siempre, bien con el valor suministrado, o con el establecido por defecto en la definición.

El segundo calificador en nuestra declaración del atributo IMG era **#IMPLIED**, que es el tercero de una terna (**#REQUIRED**, **#FIXED** e **#IMPLIED**), cuyo significado es el que vemos en la Tabla 4 (el símbolo # significa que lo que sigue es una palabra reservada).

Clave	Descripción
#IMPLIED	Opcional
#REQUIRED	Obligatorio
#FIXED "valor"	Fijo e igual a "valor"

Tabla 4. Calificadores

Si un atributo es calificado como **#REQUIRED**, significa que su inclusión es obligatoria para el *parser*, que, en caso contrario, producirá un error en el análisis. Si el atributo se califica como **#FIXED**, significa que el valor será siempre el mismo, ya se suministre como dato ó como valor por defecto. Finalmente, todos los atributos que no se califican como uno de los dos anteriores, deben ir calificados como **#IMPLIED**, ya que además, se trata de calificadores mutuamente excluyentes (sólo se admite uno por atributo).

Los espacios de nombres (*namespaces*)

Imagine el lector que tenemos que referirnos a un mismo concepto aplicado a entidades diferentes. Por ejemplo, la etiqueta <SALIDA> puede usarse para describir el punto de partida de una carrera ciclista, pero también tiene sentido como hora de salida de un empleado, o para especificar la puerta de salida obligatoria en un edificio. Para evitar posibles colisiones conceptuales, se establecieron los espacios de nombres (*namespaces*), cuyo propósito es definir un identificador único que pueda ser usado sin ambigüedad para preceder una etiqueta, permitiendo nombres iguales, con significados distintos.

Ese identificador único, adoptará la forma de *URI (Universal Resource Identifier)*, que la mayoría de las veces, asociamos a una dirección de Internet, aunque de forma alternativa, podríamos utilizar direcciones de correo electrónico, o un CLSID (identificador de clase) o cualquier otro identificador único.

Cada espacio de nombres, deberá estar ligado a un identificador diferente y la notación utilizada en XML es el atributo `xmlns`. En el Código Fuente 18 tenemos un ejemplo, para definir un espacio de nombres para la etiqueta <SALIDA> y asociarlo a un identificador único.

```
<SALIDA xmlns="http://www.eidos.es">
(...el espacio de nombres se aplica a la etiqueta y a todo su contenido, como si
cada nueva etiqueta contenida en ella, tuviese un prefijo distintivo)
</SALIDA>
```

Código Fuente 18. Definición de un espacio de nombres aplicable a la etiqueta <SALIDA>

No obstante, muchas veces lo que interesa es definir prefijos específicos a utilizar dentro del mismo documento, para distinguir posibles etiquetas iguales, pero de significados distintos. De esa forma, podríamos definir varios *namespaces* y utilizarlos como se puede apreciar en el Código Fuente 19.

```
<DOCUMENTO xmlns:CICLISMO="http://www.eidos.es/CICLISMO/"
xmlns:EMPLEADOS="http://www.eidos.es/EMPLEADOS/"
xmlns:EDIFICIO="http://www.eidos.es/EDIFICIO/">
<CICLISMO:SALIDA>Pº Castellana 100</CICLISMO:SALIDA>
<EMPLEADOS:SALIDA>18:00 Horas</EMPLEADOS:SALIDA>
<EDIFICIO:SALIDA>Puerta Principal</EDIFICIO:SALIDA>
</DOCUMENTO>
```

Código Fuente 19. Uso de prefijos de nombres aplicables a la etiqueta <SALIDA>

Conviene recordar, además, que las URI especificadas, no tienen porque ser direcciones reales de Internet, y basta con que cumplan su función como identificadores únicos. A tal efecto podríamos haber utilizado (por ejemplo) las 25 primeras cifras del número PI, o cualquier otra cosa.

Resumiendo, siempre que aparezca una etiqueta precedida de un prefijo de espacio de nombres, esto nos indica que es el poseedor del espacio de nombres el que contiene la definición de cómo hay que interpretar esa etiqueta.

Uso de espacios en blanco

También es posible establecer cuál debe ser el tratamiento que el analizador de a los espacios en blanco (espacios, tabulaciones, retornos de carro, etc.). Para ello utilizamos el atributo reservado `xml:space`, que puede valer `default` o `preserve` respectivamente según queramos ignorar o preservar los espacios en blanco. Una declaración de este tipo que optara por preservarlos tendría la forma del Código Fuente 20.

```
<!ATTLIST Titulo xml:space (default | preserve) "preserve" >
```

Código Fuente 20. Declaración de un atributo que preserva los espacios en blanco

Existen algunas palabras reservadas adicionales de menor utilización, que el lector puede consultar en el Apéndice B de ésta obra (versión en castellano) o bien en su versión inglesa original de la especificación oficial del W3C en la dirección <http://www.w3.org/TR/1998/REC-xml-19980210>.

De acuerdo con lo anterior, vamos a construir un ejemplo más completo de utilización que reúna varias de éstas declaraciones. Asignamos un atributo `GRAFICO` al elemento `Cliente` (que podría corresponder al logotipo de la empresa de ese cliente), y además, para la gestión automatizada de nuestra propia empresa, declaramos una entidad (`ENTITY`) con el nombre `LogoEidos`, y una referencia a la ubicación de dicho fichero gráfico ("`Graficos/LogoEIDOS.gif`"). Para asegurarnos de que la aplicación sabe que aplicación utilizar para manejar ese fichero, añadimos una declaración `NOTATION` indicando que la aplicación por defecto para leer el fichero, es `PaintBrush (PAINT.EXE)`.

Adicionalmente, añadimos una declaración `xml:space` para preservar el tratamiento de los espacios en blanco dentro del elemento `IdCliente`. En el Código Fuente 21 vemos el aspecto que tendría el fichero final.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clientes [
<!ELEMENT Clientes (Cliente*)>
<!ELEMENT Cliente (IdCliente, Empresa, Contacto, Ciudad, Provincia)>
<!ATTLIST Cliente GRAFICO CDATA #REQUIRED>
<!ENTITY LogoEidos SYSTEM "Graficos/LogoEIDOS.gif" NDATA GIF>
<!NOTATION GIF SYSTEM "PAINT.EXE">
<!ELEMENT IdCliente (#PCDATA)>
<!ATTLIST IdCliente xml:space (default | preserve) "preserve">
<!ELEMENT Empresa (#PCDATA)>
<!ELEMENT Contacto (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
<!ELEMENT Provincia (#PCDATA)>
]>
<Clientes>
  <Cliente GRAFICO="LogoEidos">
    <IdCliente>1</IdCliente>
    <Empresa>Defensa Personal "Orient"</Empresa>
    <Contacto>Alfonso Papo</Contacto>
    <Ciudad>Ceuta</Ciudad>
    <Pais>Ceuta</Pais>
  </Cliente>
  <Cliente GRAFICO="Graficos/Logo2.gif">
    <IdCliente>2</IdCliente>
```

```

        <Empresa>Mística y Salud S.A.</Empresa>
        <Contacto>Ana Coreta</Contacto>
        <Ciudad>Santander</Ciudad>
        <Pais>Santander</Pais>
    </Cliente>
    <Cliente GRAFICO="Graficos/Logo3.gif">
        <IdCliente>3</IdCliente>
        <Empresa>Gráficas Cadillas Inc.</Empresa>
        <Contacto>Tony Cono</Contacto>
        <Ciudad>Terrassa</Ciudad>
        <Pais>Barcelona</Pais>
    </Cliente>
</Clientes>

```

Código Fuente 21. Fichero XML completo con el DTD original modificado

Notese que en el caso del atributo GRAFICO del primer Cliente, hemos indicado exclusivamente “LogoEidos”, el nombre de una Entidad declarada en el DTD. El intérprete, deberá de sustituir en tiempo de ejecución esa referencia por el *path* real de ese fichero, “Graficos/LogoEIDOS.gif”.

DTD externos

En muchas ocasiones, un grupo de ficheros XML comparte un mismo DTD, con lo que lo más apropiado es situar el fichero de declaraciones aparte y hacer referencia a él mediante una declaración SYSTEM o PUBLIC dentro de cada fichero de datos. En éste caso, las declaraciones del Código Fuente 22 son ambas válidas:

```

<!DOCTYPE NombreDocumento PUBLIC "Identificador_público" "Ubicación_Fichero" >
<!DOCTYPE NombreDocumento SYSTEM "Ubicación_Fichero" >

```

Código Fuente 22. Declaración de un DTD externo

En el primer caso, "*Identificador_Público*" hace referencia a una cadena de texto que el analizador tratará de buscar para analizar el DTD dentro de un recurso común del sistema o repositorio externo. En el segundo, el analizador buscará directamente la "*Ubicación_Fichero*" para procesarlo.

Ejercicios

1. ¿Qué son los espacios de nombres y cuál es el problema que resuelven dentro de los DTD?
2. ¿Qué declaración utilizamos para indicar que el contenido de un elemento es texto plano?
3. ¿Qué significa CDATA?
4. ¿Qué definimos al declarar un elemento como ENTITY?
5. ¿Mediante qué declaración convertimos un elemento en una entidad programable?
6. ¿Qué mecanismo interviene en la gestión y definición de espacios en blanco?

7. ¿Cómo expresamos que un elemento **debe** tener un valor obligatoriamente?
8. ¿Un DTD puede residir fuera del documento XML? Si es así, ¿cómo se utiliza?
9. Crea el DTD correspondiente a un fichero de datos que definiera valores relativos al funcionamiento de un semáforo

Otros elementos autodescriptivos

Resource Description Framework (RDF)

Dentro de los mecanismos de descripción que se encuentran en fase de estudio actual por el W3C, destacan dos, que pretenden, por un lado simplificar, y por otro completar, las capacidades de definición que hemos estudiado con relación a los DTD. Estas dos propuestas en fase de estudio se denominan XML-Schemas y Resource Description Framework (RDF).

Comenzaremos por ésta última debido a que no vamos a extendernos en su explicación dentro del contexto de estos artículos, ya que se trata de una propuesta en fase de desarrollo, de la que no existe todavía un soporte de software universalmente aceptado, como sucede (a la fecha) con XML-Schemas.

RDF es la más ambiciosa de las propuestas del W3C con relación a los metadatos. Se trata de una sintaxis para la descripción de recursos, donde se entiende por recurso, todo aquello que puede designarse mediante una URI (*Universal Resource Identifier*). La terminología es bastante teórica, pero simplificando podríamos decir que RDF es un modelo para hablar acerca de temas diversos. Aquellas cosas sobre las que discutimos, o que usamos o a las que nos referimos en un esquema RDF, se llaman recursos.

RDF utiliza como elementos constructores 3 componentes básicos: Recursos, Propiedades y Sentencias. Por propiedades, entendemos todo atributo significativo y específico de un recurso. Por sentencias, una combinación de un recurso, una propiedad y un valor de la propiedad.

Cada propiedad tiene un rango (el conjunto de valores que puede describir) y un dominio: la clase a la que se aplican esos valores. Posiblemente veremos pronto una proposición final de RDF y un soporte de ésta proposición por medio de los fabricantes de software.

Si, como parece, RDF cobra la importancia que desean sus mentores y esto se ve reflejado en una aceptación por parte de los fabricantes, dedicaremos un artículo a explicar su funcionamiento. Por ahora, vamos a centrarnos en la otra proposición (*XML-Schemas*) soportada por los dos navegadores más populares, Internet Explorer 5.0 y por la próxima versión de Netscape 5.0, según hemos podido comprobar utilizando la beta disponible del producto.

XML-Schemas

La última propuesta -en fase de terminación por el W3C- de XML-Schemas data del 17/12/99. Eso significa que estamos hablando de una tecnología que se encuentra todavía en fase de desarrollo, si bien, según la propia W3C se espera una recomendación final y definitiva para Marzo/2000. Aún así, varios productos comerciales ya soportan diferentes grados de implementación del estándar, como hemos comentado, y lo que es más importante, el modelo de objetos de ADO. Todo ello supone un tremendo avance sobre los DTD, especialmente si tenemos en cuenta la Tabla 5 que enumera las mejoras respecto a los DTD.

Mejoras de XML-Schemas	Descripción
Tipos de datos	Ahora hay 37 tipos definibles más la posibilidad de definir tipos propios y una representación léxica (basada en máscaras).
Escritos en XML	Habilita el uso de analizadores XML (parsers)
Orientadas a objetos	Permite extender o restringir tipos de datos basándose en otros ya definidos.
Permite expresar conjuntos	Los elementos hijos pueden aparecer en cualquier orden
Especificación de contenido único	Se pueden establecer claves de contenido y regiones de unicidad.
Solución implícita a los namespaces	Se pueden definir múltiples elementos con el mismo nombre y contenidos distintos.
Contenidos especiales	Se pueden definir elementos con contenido nulo.
Equivalencia de clases	Se pueden definir clases equivalentes para ser utilizadas indistintamente.

Tabla 5. Mejoras respecto a los DTD

Al igual que los DTD, los Schemas tienen por objeto definir la especificación formal de los elementos y atributos que están permitidos dentro de un documento XML y cuáles son las relaciones existentes entre ellos.

En general, los elementos integrantes de un XML-Schema se definen mediante las etiquetas que aparecen en la Tabla 6.

ELEMENTO	DESCRIPCIÓN
Attribute	Hace referencia a un atributo declarado anteriormente mediante una etiqueta AttributeType
AttributeType	Define un tipo de atributo para usarse dentro de un elemento del Schema
Datatype	Especifica el tipo de dato de un ElementType o AttributeType
Description	Suministra información adicional sobre un ElementType o AttributeType
Element	Hace referencia a un elemento declarado anteriormente mediante una etiqueta ElementType
ElementType	Define un tipo de elemento para usarlo como elemento del Schema
Group	Organiza contenidos en un grupo para especificar una secuencia
Schema	Identifica el comienzo de una declaración de Schema

Tabla 6. Etiquetas de los elementos que integran un XML-Schema

Explicaremos el funcionamiento de cada palabra reservada, y comenzaremos con el equivalente a la declaración del DTD dentro de este contexto: <Schema>. Dispone de atributos para definir sus características adicionales, tales como su nombre (name), y los espacios de nombres (namespaces) que utilizará como estándares. Una declaración típica podría tomar la forma del Código Fuente 23

```
<Schema name="Clientes" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
```

Código Fuente 23. Definición de un Schema

Definimos aquí un *Schema* para la tabla de Clientes, utilizando dos espacios de nombres: uno para el *Schema* en sí, y otro para los tipos de datos. Más adelante, en el ejemplo veremos más detalles de su utilización.

Las formas equivalentes a las declaraciones ELEMENT y ATTLIST de un DTD en *XML-Schemas* son, respectivamente, Element y Attribute, si bien para poder definir elementos o atributos mediante estas declaraciones es preciso haberlas definido primero, usando las declaraciones ElementType y AttributeType. Cada una de ellas dispone de atributos para establecer las características que usábamos con sus equivalentes DTD, más otras nuevas que no están disponibles en éstos últimos. Veamos un primer ejemplo de formatos equivalentes. Si partimos de una definición reducida de nuestra lista de clientes como hacemos en el Código Fuente 24, una primera transformación (necesitaremos más, pero vamos por partes), sería la del Código Fuente 25.

```
<!DOCTYPE Clientes [
  <!ELEMENT Clientes (Cliente+)>
```

```

<!ELEMENT Cliente (IdCliente, Empresa, Contacto, Ciudad, Provincia)>
<!ELEMENT IdCliente (#PCDATA)>
<!ELEMENT Empresa (#PCDATA)>
<!ELEMENT Contacto (#PCDATA)>
<!ELEMENT Ciudad (#PCDATA)>
<!ELEMENT Provincia (#PCDATA)>
]>

```

Código Fuente 24. Definición básica del DTD para la Lista de Clientes

```

<?xml version="1.0" encoding="UTF-8"?>
<Schema name="Clientes"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">

  <!-- Primero definimos los elementos de mas bajo nivel -->

  <ElementType name="IdCliente" />
  <ElementType name="Empresa" />
  <ElementType name="Contacto" />
  <ElementType name="Ciudad" />
  <ElementType name="Provincia" />

  <!-- Despues el elemento contenedor de todos ellos (Cliente) -->

  <ElementType name="Cliente" content="eltOnly" model="closed" order="seq">

    <element type="IdCliente" />
    <element type="Empresa" />
    <element type="Contacto" />
    <element type="Ciudad" />
    <element type="Provincia" />
    <AttributeType name="Logotipo" />
    <attribute type="Logotipo" />

  </ElementType>

  <!-- Finalmente, definimos el elemento Clientes, compuesto de elementos Cliente -->
  <ElementType name="Clientes" content="eltOnly" model="closed" order="seq">

    <element type="Cliente" minOccurs="0" maxOccurs="*" />

  </ElementType>

</Schema>

```

Código Fuente 25. 1ª Fase de la transformación del DTD en el Schema equivalente

Los elementos y atributos de un *Schema* se definen mediante etiquetas **ElementType** y **AttributeType** que suministran la definición y el tipo de elemento o atributo. Una vez definidos, cada instancia de un elemento o atributo se declara usando las etiquetas **<element.. >** y **<attribute.. >**, de forma similar a cuando definimos un tipo de datos de usuario en C++ o Visual Basic (por ejemplo, mediante *typedef*), y más tarde declaramos variables de ese tipo. Además, igual que sucede en éste caso, si queremos hacer referencia a los tipos declarados, deberemos hacerlo en instrucciones posteriores a la declaración (no antes), ya que el analizador no los encontrará.

Hasta aquí, nos hemos limitado a crear la estructura básica, el esqueleto equivalente a la definición de que disponíamos en el DTD anterior, sin añadir características nuevas. Conviene llamar la atención sobre los puntos básicos (observe el lector que estamos usando sintaxis XML, a diferencia de los

DTD): El propio *Schema* se define en la primera etiqueta, tal y como habíamos comentado anteriormente. A continuación vamos definiendo los elementos, por así decirlo, desde el más anidado hasta el más superficial, según la norma que comentábamos anteriormente: vamos creando los *ladrillos* individuales y -con ellos- vamos definiendo los componentes más complejos.

Sin embargo, no quiero concluir ésta entrega sin comentar aspectos de la sintaxis, que resultarán desconocidos para el lector. Aparte de la forma de construcción de la estructura -de dentro afuera-, en primer lugar, vemos que se definen dos espacios de nombres dentro de la etiqueta *Schema*: uno general (`xmlns="urn:schemas-microsoft-com:xml-data"`) y otro específico para los tipos de datos (`xmlns:dt="urn:schemas-microsoft-com:datatypes"`). Ambos son valores únicos de recursos predefinidos (URI), pero podríamos usar cualquier otro que cumpliera la condición de unicidad exigida en los espacios de nombres. La razón de ésta definición, es disponer de una especificación general para los tipos de datos, que permita al usuario definir los suyos propios independientemente del estándar.

En segundo término, revisemos los atributos utilizados en las definiciones de *Cliente* Y *Clientes*. Para *Cliente*, usamos cuatro atributos: el nombre (**name**="Cliente"), el contenido (**content**="eltOnly"), que nos indica que sólo puede albergar otros elementos, el modelo (**model**="closed"), que nos dice que sólo pueden aparecer los elementos citados y no otros, y el orden de aparición (**order**="seq"), que especifica que los elementos que aparezcan deberán hacerlo en el mismo orden secuencial que se indica en el *Schema*. Resumiendo, *Cliente* sólo puede contener otros elementos (y no datos en sí), los elementos que puede contener serán siempre los mismos y además aparecerán en el orden especificado.

Además, existe un atributo en la declaración de los elementos *Cliente*: *Logotipo*. Aquí es preciso tener muy en cuenta *el lugar* en que se ha definido el atributo. El lugar sirve para establecer el ámbito, y como podemos ver, *AttributeType* se define dentro de la estructura del elemento *Cliente*, lo que significa que sólo es aplicable a *Cliente*, y podríamos utilizar el mismo nombre (*Logotipo*) para definir otro atributo externo y utilizarlo con otro elemento. Esa es la forma en que el ámbito determina el contexto de utilización de un atributo. A continuación, se usa la etiqueta `<attribute... >` para declarar el atributo *Logotipo*, una vez definido dentro de su contexto.

Para concluir la descripción, el elemento *Clientes*, que dispone de los mismo atributos básicos que el elemento *Cliente*, contiene solamente un elemento: *Cliente*, definido previamente, y dos atributos: **minOccurs** = "0" y **maxOccurs** = "*", que indican la cardinalidad del elemento. *Cliente* puede aparecer un mínimo de 0 veces y un máximo no determinado.

En el Código Fuente 26 aparece la sintaxis completa de *ElementType* tal y como aparece en la especificación (el lector puede utilizar igualmente las ayudas de MSDN que se incluyen con Visual Studio, donde hay referencias a XML).

```
<ElementType
  content="{empty | textOnly | eltOnly | mixed}"
  dt:type="datatype"
  model="{open | closed}"
  name="idref"
  order="{one | seq | many}" >
```

Código Fuente 26. Sintaxis de *ElementType*

Nos falta establecer atributos adicionales para los elementos, de forma que podamos aprovechar mejor las características de XML-Schemas, y conseguir una definición más precisa de los elementos,

consiguiendo una mejora en el rigor de las definiciones y una reducción de los posibles errores en los datos.

Ejercicios

1. ¿En qué consiste la especificación RDF?
2. ¿Cuáles son los equivalentes a ELEMENT y ATTRIBUTE en XML-Schemas?
3. ¿Qué se declara primero, **element** o **ElementType**? ¿Por qué?
4. ¿Cuál debe ser la ubicación de **AttributeType** para indicar que sólo actúa sobre un elemento y no sobre todos ellos?
5. ¿Qué especificamos mediante **Datatype**?
6. ¿Y mediante **Model**?
7. ¿Cómo se indica que un elemento sólo puede contener otros elementos?
8. ¿Cómo se expresa el número de veces que puede aparecer un elemento?

XML-Schemas

Continuamos el estudio de la especificación de XML-Schemas y su uso en las definiciones estructurales de documentos XML. En el artículo anterior, planteábamos una reconversión de nuestro ejemplo de definición de una lista de libros mediante una DTD, para expresarla utilizando XML-Schemas. Habíamos definido un esqueleto, o estructura de la definición y, a través de él, revisábamos la especificación, adaptándola al ejemplo. También, recordamos la definición sintáctica de la etiqueta `<ElementType>`, en el Código Fuente 27.

```
<ElementType
  content="{empty | textOnly | eltOnly | mixed}"
  dt:type="datatype"
  model="{open | closed}"
  name="idref"
  order="{one | seq | many}" >
```

Código Fuente 27. Sintaxis de ElementType

Donde podemos ver que existen hasta cinco atributos posibles; los cuatro ya citados, más el atributo `dt:type` que permite indicar el tipo de dato que va a admitir el elemento. Las posibilidades del primer atributo, `content`, se expresan mediante los valores preestablecidos que figuran en la definición: "empty" si el elemento está vacío, "textOnly", si sólo admite texto, "eltOnly", si sólo admite otros elementos y "mixed" en el caso de que admita tanto elementos como texto.

Model, indica si es posible incluir elementos no definidos (mediante el valor "open"), o si la estructura está cerrada y sólo se puede usar los establecidos en el Schema (valor "closed"). Name, es, por

supuesto, el nombre del elemento, y order, establece la forma en que deben aparecer los elementos: Sólo uno de cada elemento definido ("one"); todos, tal y como están en la definición ("seq"), o bien, aparición en cualquier orden ("many"), si es que aparece alguno.

Con relación a los tipos de datos (dt:type), se trata de la característica que más se ha potenciado en XML-Schemas, respecto a los DTD. No sólo disponemos de una gama amplia de posibilidades de definición de tipos, sino que podemos construir nuestros propios tipos de usuario, a partir de los existentes en la especificación. En la Tabla 7 incluimos los tipos de datos que pueden declararse.

Tipo de Dato	Descripción
bin.base64	MIME-style Base64 Objetos binarios BLOB.
bin.hex	Dígitos Hexadecimales representando octetos.
Boolean	0 ó 1, donde 0 = "false" y 1 = "true".
Char	Cadena, de un carácter de longitud
Date	Fecha en formato ISO 8601 sin datos de hora. Por ejemplo: "2000-01-05".
dateTime	Fecha en formato ISO 8601 con datos de hora opcionales y zona horaria no opcional. Las fracciones de segundos se pueden precisar hasta el nanosegundo. Por ejemplo: "1988-04-07T18:39:09".
dateTime.tz	Fecha en formato ISO 8601 con datos de hora opcionales y zona horaria no opcional. Las fracciones de segundos se pueden precisar hasta el nanosegundo. Por ejemplo: "1988-04-07T18:39:09-08:00".
fixed.14.4	Lo mismo que Number, pero hasta 14 dígitos a la izquierda del punto decimal, y no más de 4 a la derecha.
Flota	Número, sin límite de dígitos; puede llevar signo, parte decimal y opcionalmente, un exponente. La puntuación en formato Inglés Americano. (Entre 1.7976931348623157E+308 a 2.2250738585072014E-308.)
Int	Número, con signo opcional, sin parte decimal y sin exponente.
Number	Número, sin límite de dígitos; puede llevar signo, parte decimal y opcionalmente, un exponente. La puntuación en formato Inglés Americano. (Entre 1.7976931348623157E+308 a 2.2250738585072014E-308.)
Time	Subconjunto del formato ISO 8601, sin fecha, ni marca zonal. Por ejemplo: "08:15:27".
time.tz	Subconjunto del formato ISO 8601, sin fecha, pero con marca zonal opcional. Por ejemplo: "08:15:27-05:00".
i1	Entero representado con un byte. Número, con signo opcional, sin parte decimal y sin exponente. Por ejemplo: "1, 127, -128".
i2	Entero representado con una palabra (word). Número, con signo opcional, sin parte decimal y sin exponente. Por ejemplo: "1, 702, -22768".

	sin parte decimal y sin exponente. Por ejemplo: "1, 703, -32768".
i4	Entero representado con 4 bytes. Número, con signo opcional, sin parte decimal y sin exponente. Por ejemplo "1, 703, -32768, 148343, -1000000000".
R4	Número, sin límite de dígitos; puede llevar signo, parte decimal y opcionalmente, un exponente. La puntuación en formato Inglés Americano. (Entre 3.40282347E+38F y 1.17549435E-38F).
R8	Igual que "float." Número, sin límite de dígitos; puede llevar signo, parte decimal y opcionalmente, un exponente. La puntuación en formato Inglés Americano. (Entre 1.7976931348623157E+308 y 2.2250738585072014E-308).
ui1	Entero sin signo. Número, sin signo, sin parte decimal y sin exponente. Por ejemplo: "1, 255".
ui2	Entero sin signo, 2 bytes. Número, sin signo, sin parte decimal y sin exponente. Por ejemplo: "1, 255, 65535".
ui4	Entero sin signo, 4 bytes. Número, sin signo, sin parte decimal y sin exponente. Por ejemplo: "1, 703, 3000000000".
Uri	Identificador Universal de Recursos. (Universal Resource Identifier (URI)). Por ejemplo: "urn:schemas-microsoft-com:Office9".
Huid	Identificador Único Universal (Universal Unique Identifier) Dígitos Hexadecimales representando octetos, opcionalmente formateados mediante guiones que se ignoran. Por ejemplo: "333C7BC4-460F-11D0-BC04-0080C7055A83".

Tabla 7. Tipos de datos

Como vemos, la variedad de tipos de datos va desde los objetos binarios (BLOB) con los que pueden hacerse referencia a ficheros de sonido, o gráficos, hasta los identificadores universales, utilizados, por ejemplo en la tecnología ActiveX.

Además la Recomendación XML 1.0 del W3C define tipos enumerados (notaciones y enumeraciones) y un conjunto de tipos *tokenized*. Se denominan *primitivas* dentro de la documentación.

La Tabla 8 nos muestra la lista de tipos primitivos, según se definen en la Sección 3.3.1 de la recomendación.

En cuanto a la etiqueta para la definición de atributos, AttributeType se define mediante la sintaxis que aparece en el código Fuente 28.

Tipo Primitivo	Entidad XML representada
Entity	Representa a ENTITY

Entities	Representa a ENTITIES
Enumeration	Representa un tipo enumerado (sólo soportado por atributos)
Id	Representa a ID
Idref	Representa a IDREF
Idrefs	Representa a IDREFS
Nmtoken	Representa a NMTOKEN
nmtokens	Representa a NMTOKENS
Notation	Representa a NOTATION
String	Representa a string (cadena de caracteres de longitud variable)

Tabla 8. Tipos Primitivos

```
<AttributeType
  default="valor por defecto"
  dt:type="tipo predefinido"
  dt:values="enumeración de valores posibles"
  name="idref"
  required="{yes | no}" >
```

Código Fuente 28. Sintaxis de AttributeType

Como el lector podrá deducir de la sintaxis, el primer atributo de AttributeType, default, especifica el valor por defecto que tendrá el atributo de no indicarse ninguno. El segundo, dt:type, corresponderá con uno de los tipos de datos predefinidos en la tabla anterior, y el tercero, dt:values se usará en los casos en los que el valor a adoptar sólo pueda ser uno de los que se especifican en la lista de valores enumerados, separados por comas. El nombre, name, será como siempre un valor único, y finalmente, el atributo required indicará si el atributo es obligatorio o no. Con estas especificaciones lo que hacemos es DEFINIR los elementos y atributos que vamos a DECLARAR posteriormente. En las declaraciones, también disponemos de una cierta flexibilidad, admitida por el lenguaje. En concreto la sintaxis de las etiquetas <element...> y <attribute...> es la que aparece en el Código Fuente 29.

```
<element type="tipo de elemento" [minOccurs="{0 | 1}"]
[maxOccurs="{1 | *}"] >

<attribute default="valor por defecto" type="tipo de atributo" [required="{yes |
no}"] >
```

Código Fuente 29. Sintaxis de <element...> y <attribute...>

Donde los valores de los atributos han sido ya comentados con anterioridad. Además de estas etiquetas, es posible utilizar una etiqueta adicional <group...>, que permite establecer agrupaciones de elementos y la forma en que éstas agrupaciones deben de aparecer, exactamente como lo hace el atributo order de un elemento. Su función es similar a la de los bloques de código en Transact-SQL

donde agrupamos un conjunto de sentencias en un bloque mediante BEGIN/END. En el Código Fuente 30 se nos muestra la sintaxis de Group.

```
<group maxOccurs="{1 | *}" minOccurs="{0 | 1}"
order="{one | seq | many}" >
```

Código Fuente 30. Sintaxis de <group...>

Donde, siguiendo las convenciones sintácticas comentadas hasta ahora, el atributo order de group, tiene el mismo significado que el atributo order de un elemento <element>, si bien, de existir éste último, lo establecido por él tiene precedencia. Fijémonos Código Fuente 31, tomado del MSDN de Microsoft.

```
<ElementType name="x" order="one">
  <group order="seq">
    <element type="x1">
    <element type="y1">
  </group>
  <group order="seq">
    <element type="x2">
    <element type="y2">
  </group>
</ElementType>
```

Código Fuente 31. Ejemplo de uso de <group...> en una definición <element...>

Si tenemos una definición de tipo, podemos considerar una instancia válida de la definición anterior, los ejemplos que aparecen en el Código Fuente 32.

```
<x>
  <x1/>
  <y1/>
</x>

y también:

<x>
  <x2/>
  <y2/>
</x>
```

Código Fuente 32. Ejemplo de datos válidos a partir de la definición anterior

Observe el lector que en la definición del elemento "x", se establece como atributo order = "one", y que disponemos de dos grupos, que deben aparecer en el orden en que se definen: uno formado por (X1, Y1) y el otro formado por (X2, Y2). Como order="one", obliga a que sólo se instancie uno de la lista, los dos ejemplos son válidos.

Según las definiciones, podemos modificar nuestro Schema en una segunda fase para ajustarlo más a nuestras necesidades (Código Fuente 33).

```
<?xml version='1.0'?>
<Schema name="Clientes"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">

  <!-- Primero definimos los elementos de mas bajo nivel -->

  <ElementType name="IdCliente" />
  <ElementType name="Empresa" />
  <ElementType name="Contacto" />
  <ElementType name="Ciudad" />
  <ElementType name="Provincia" />

  <!-- Despues el elemento contenedor de todos ellos (Cliente) -->

  <ElementType name="Cliente" content="eltOnly" model="closed" order="seq">

    <element type="IdCliente" dt:type="int" />
    <element type="Empresa" dt:type="string"/>
    <element type="Contacto" dt:type="string"/>
    <element type="Ciudad" dt:type="string"/>
    <element type="Provincia" dt:type="string"/>
    <AttributeType name="Logotipo" dt:type="string" required="No" />
  <attribute type="Logotipo" />

</ElementType>

  <!-- Finalmente, definimos el elemento Clientes, compuesto de elementos Cliente -->
  <ElementType name="Clientes" content="eltOnly" model="closed" order="seq">

    <element type="Cliente" minOccurs="0" maxOccurs="*" />

  </ElementType>

</Schema>
```

Código Fuente 33. 2ª Fase de la transformación del DTD en el Schema equivalente

El lector puede comprobar la validez de la sintaxis de la declaración de este esquema, creando un fichero de texto con la extensión XML e incluyendo el esquema anterior. Bajo IE5, la salida tendrá el aspecto que aparece en la figura 3.

Hasta aquí, hemos visto la forma en que podemos definir los datos. Pero recordemos que XML se distingue precisamente por la separación entre los datos y la presentación. Por ello, en el tema siguiente abordaremos la presentación mediante Hojas de Estilo Extendidas (XSL), que mezcla, por un lado la capacidad de selección de elementos y por otro, las capacidades de formato de la salida.

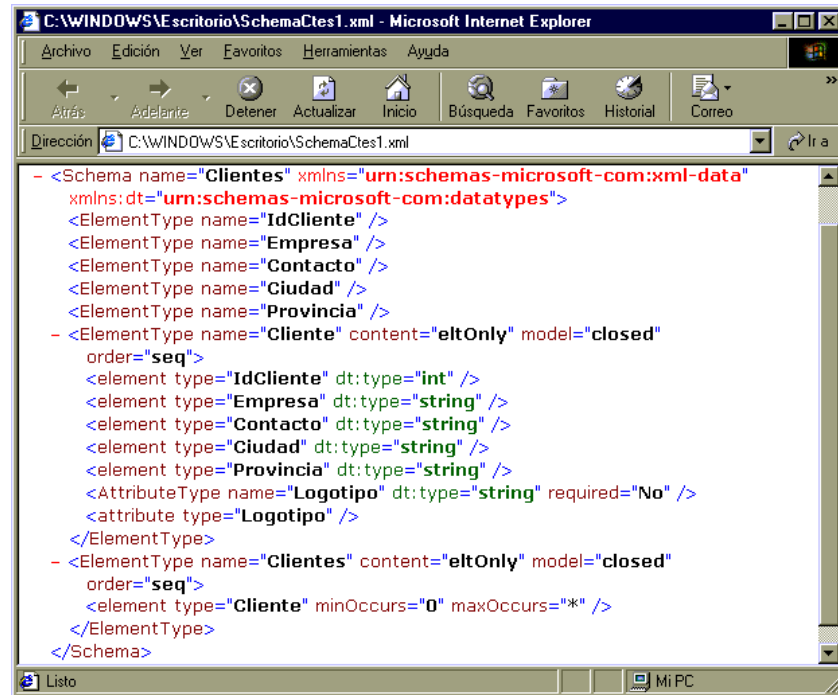


Figura 3. Interpretación del Schema anterior por Explorer 5.0

Ejercicios

1. ¿Qué significa el atributo **mixed**?
2. ¿Qué sentido tiene la utilización de **group**?
3. ¿Qué se consigue mediante el atributo **order**?
4. ¿Qué representa **enumeration**?
5. ¿Qué tipos predefinidos se usan para el almacenamiento de fechas?
6. ¿Cómo se gestiona la cardinalidad en XML-Schemas?
7. Crea el esquema equivalente al DTD de la casa que vimos en capítulos anteriores

Mecanismos de presentación: hojas de estilo (CSS) y el lenguaje de hojas estilo extendidas: XSL

Al ser XML un lenguaje de marcas que permite la autodefinición de los datos, nos hemos centrado en sus características y en esos mecanismos de definición, especialmente en los DTD y XML-Schemas. Pero, recordemos que, una de las ventajas principales que mencionábamos, era, precisamente, la capacidad de separar los datos, de la forma en que esos datos son mostrados (ya sea utilizando un navegador, o una herramienta de desarrollo, o cualquier otro mecanismo).

Mediante la separación de la forma de presentación, se consiguen importantes ventajas: podemos disponer de un método uniforme de presentación definido una sola vez, y utilizarlo con múltiples documentos XML, o podemos definir un conjunto de modos de presentación y permitir al usuario que utilice el que más le convenga en cada caso.

Esto se consigue utilizando un lenguaje especial de presentaciones (denominado XSL o Extensible StyleSheet Language), que utiliza una sintaxis XML, -por razones de coherencia- y está definido como estándar por la W3C, igual que todos los lenguajes asociados a XML que hemos estado analizando. Si el lector está interesado en revisar la especificación completa del estándar, podrá encontrarla en la dirección Internet: <http://www.w3.org/TR/WD-xsl/>

Algo a tener en cuenta al comenzar el estudio de XSL es que éste estándar no define la presentación en sí, sino el conjunto de registros que van a visualizarse y el orden que tendrán una vez seleccionados. Por tanto, no debemos ver a XSL como un lenguaje de presentación sino más bien como un lenguaje de selección de contenidos. Como ya hemos dicho, de cara a la presentación, el estándar CSS es

suficientemente potente y permite definir cómo va a mostrarse cada elemento, con un conjunto de palabras reservadas semejante al de los estilos propios de los procesadores de texto. Lo que *no* permite CSS es la selección de *qué aparecerá* y mucho menos, puede hacer depender el formato del contenido del contenido mismo (XSL posee sentencias *if*, *choose*, etc., que permiten que la presentación varíe en función del contenido de una etiqueta o atributo).

Además, y de cara al tratamiento de datos, existe otra ventaja importante: la separación de los datos permite la reutilización dinámica de un mismo *recordset* mediante *vistas* diferentes que pueden responder a peticiones del usuario: no es necesario volver a solicitar los datos del servidor para cambiar el modo de ordenación, o para filtrar un conjunto de registros dentro del *recordset* principal. Basta con cambiar los atributos de proceso del fichero XSL y pedir al navegador que *refresque* la información, para visualizar un conjunto distinto de resultados, sin haber tenido que solicitar ese conjunto al servidor. Esto reduce en buena parte el tráfico de red, que, en el caso de aplicaciones para Internet, resulta crucial.

Respecto al tratamiento de datos, hay que destacar que los planes de desarrollo concernientes a los servidores de datos más populares incluyen opciones que permiten devolver conjuntos de datos en formato XML como respuesta a peticiones de los clientes. Tanto Oracle, como SQL-Server 2000 ó DB2 de IBM planean dicho soporte. Por ejemplo, utilizando SQL-Server 2000, un usuario puede solicitar datos desde el Explorador al servidor, y recibir como respuesta un conjunto de registros en formato XML, que el Explorador muestre como resultados en una tabla creada dinámicamente.

¿Cómo funciona XSL?

En primer lugar, y a diferencia de los ficheros de definición de datos, que pueden estar embebidos en el propio fichero XML, o bien separados de él, los ficheros de presentación están diseñados como ficheros independientes. Como apuntábamos antes, es posible diseñar un modelo de presentaciones para distintos conjuntos de datos, creando una especie de *estilo*, aplicable a distintos ficheros XML. Por el contrario, también podemos hacer un abordaje opuesto: crear diferentes estilos de presentación para similares (o idénticos) conjuntos de datos, y permitir que sea el usuario el que seleccione la forma de presentación.

Este enfoque, como habrá supuesto el lector, supone que en el fichero de datos debe existir una referencia externa al fichero XSL que almacena la presentación. (Aquí debemos ser cuidadosos: el fichero XSL debe estar situado en la misma ubicación -Directorio o URI- que el fichero de datos).

En tiempo de ejecución (o de interpretación, como se prefiera), el analizador encargado de mostrar los resultados (el programa que disponga del *parser*, que será un navegador, una herramienta de desarrollo o una utilidad de terceros), deberá de realizar varias acciones que hemos esquematizado brevemente en la figura 4.

Primero, lee la información del fichero XML, que hace las veces de punto de partida. A continuación, desglosa este fichero para buscar los metadatos: bien en forma de DTD o, como ya hemos visto, usando un Schema de XML. Una vez hecho esto, contrasta la estructura definida con los datos para verificar que el fichero está bien formado y es válido.

Si cumple ambas condiciones, crea en memoria una imagen de la información que debe de mostrar (imagen que tomará la forma de jerarquía de objetos programable, lo que será objeto de estudio en los siguientes artículos), y abre el fichero XSL de presentación. Si la estructura de éste fichero es igualmente válida, *funde ambos ficheros, recorriendo los datos* de acuerdo con las instrucciones indicadas por el fichero XSL, y produciendo una salida HTML acorde con lo indicado.

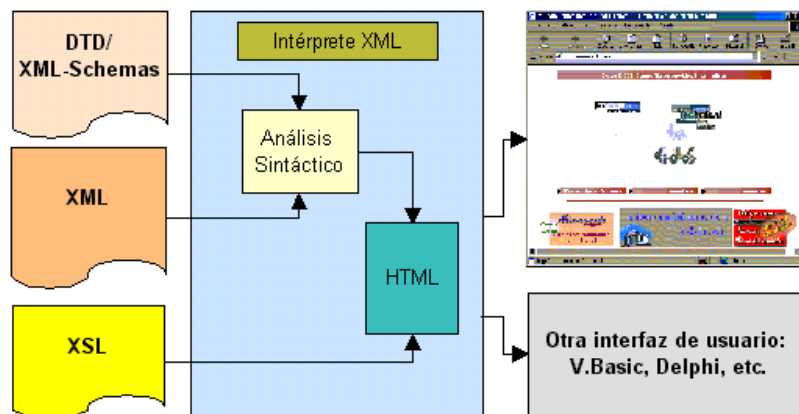


Figura 4. Esquema de interpretación de un documento XML con un DTD incluido y referencias a otros documentos XSL

Por fin, la salida HTML es mostrada mediante un mecanismo adecuado: un navegador, una interfaz de usuario de una herramienta de programación, etc. Aunque la herramienta utilizada para la interpretación tenga algo más de trabajo en éste segundo caso que en el de la utilización de HTML estándar, las ventajas son grandes, como vamos a ver.

La instrucción que hace referencia a un fichero externo XSL dentro de un fichero XML es parecida a las de otras referencias externas a ficheros.

```
<?xml:stylesheet type="text/xsl" href="Fichero_de_Formato.xsl"?>
```

Código Fuente 34. Referencia externa a un fichero XSL

Obsérvese, en el Código Fuente 34, que existe un atributo type con el valor "text/xsl". Este valor también podría ser "text/css", en el caso de que la presentación prefiramos hacerla mediante hojas de estilo en cascada (CSS).

Presentación mediante Hojas de Estilo en Cascada

Recordemos que las hojas de estilo en cascada permiten al usuario definir estilos (también llamados “clases”), que pueden aplicarse a cualquier etiqueta HTML mediante el atributo Class.

La especificación CSS³ define un conjunto de palabras reservadas para ser interpretadas por los navegadores de forma similar a como los procesadores de texto interpretan los códigos de presentación. Cuando una etiqueta HTML incluye en un atributo Class una referencia a un estilo definido por el usuario, el intérprete del navegador utiliza esa definición para generar una salida acorde con la definición. En el Código Fuente 35 vemos un ejemplo de utilización de este mecanismo.

³ Existe una especificación CSS básica o primaria y una extendida denominada CSS2, que soportan las últimas versiones de los navegadores.

```
<STYLE>
.Estilo1 {font-size:12pt.; color:green}
</STYLE>

<H1 Class="Estilo1">Texto en verde y con tamaño de 12 puntos</H1>
```

Fuente 35. Ejemplo de uso de un estilo definido por el usuario

En el caso de utilizar hojas de estilo con ficheros XML, el modelo de funcionamiento es similar, aunque no igual: El usuario debe de crear un fichero de hojas de estilo (fichero con extensión CSS), en el que se hayan definido tantos estilos como elementos se quieran mostrar con formato especial. Aquellos elementos no definidos serán mostrados según el valor estándar del navegador.

En el Código Fuente 36 vemos una definición de una hoja de estilo, en la que hemos establecido el modo de presentación para cada elemento XML de nuestro documento de prueba.

```
IdCliente { display: block; font-family: Arial; font-size: 10 pt; font-weight: bold }
Empresa { display: inline; font-family: Tahoma, Arial, sans-serif; color:
#000080;font-size: 12 pt; text-transform: capitalize; text-align:center; word-
spacing: 2; font-style: italic; font-weight: bold;margin-top: 2; margin-bottom: 2 }
Contacto { display: inline; font-family: Book Antiqua; color: #0000FF; font-size:
14 pt; text-decoration: underline; font-style: italic }
Ciudad { display: inline; font-family: Arial; font-size: 12 pt; color:
#800000;font-variant: small-caps; border-style: solid }
Provincia { display: inline; font-size: 12 pt; color: #0000FF; font-weight:
bold;background-color: #C0C0C0; border-style: solid }
```

Código Fuente 36. Definición de una hoja de estilo para utilizar con la lista de Clientes

Si aplicamos ésta definición a un fichero XML que contenga datos, de acuerdo con la definición establecida, como puede ser el que mostramos en el Código Fuente 37, obtendremos en el navegador la salida formateada que aparece en la Figura 5.

```
<?xml version="1.0"?>
<!DOCTYPE Clientes [
  <!ELEMENT Clientes (Cliente*)>
  <!ELEMENT Cliente (IdCliente, Empresa, Contacto, Ciudad, Pais)>
  <!ELEMENT IdCliente (#PCDATA)>
  <!ELEMENT Empresa (#PCDATA)>
  <!ELEMENT Contacto (#PCDATA)>
  <!ELEMENT Ciudad (#PCDATA)>
  <!ELEMENT Pais (#PCDATA)>
]>
<?xml:stylesheet type="text/css" href="ListaClientes.css"?>
<Clientes>
  <Cliente>
    <IdCliente>1</IdCliente>
    <Empresa>Defensa Personal "Orient"</Empresa>
    <Contacto>Alfonso Papo</Contacto>
    <Ciudad>Ceuta</Ciudad>
    <Pais>Ceuta</Pais>
  </Cliente>
  <Cliente>
    <IdCliente>2</IdCliente>
    <Empresa>Mística y Salud S.A.</Empresa>
    <Contacto>Ana Coreta</Contacto>
```

```

        <Ciudad>Santander</Ciudad>
        <Pais>Santander</Pais>
    </Cliente>
    <Cliente>
        <IdCliente>3</IdCliente>
        <Empresa>Gráficas Cadillas Inc.</Empresa>
        <Contacto>Tony Cono</Contacto>
        <Ciudad>Terrassa</Ciudad>
        <Pais>Barcelona</Pais>
    </Cliente>
</Clientes>

```

Código Fuente 37. Contenido del fichero XML con datos de una hipotética lista de libros

Observe el lector la línea donde se hace la referencia externa al fichero CSS, (que comienza con `<?xml:stylesheet>`), porque es ahí donde el navegador carga el fichero. Además, como el CSS contiene conjuntos de valores distintos para cada elemento definido en el fichero XML, el resultado puede variar totalmente para cada elemento, y no tenemos que definirlo más que una vez: si en vez de tres registros tuviésemos 30 ó 300, el fichero CSS no necesita cambiarse.

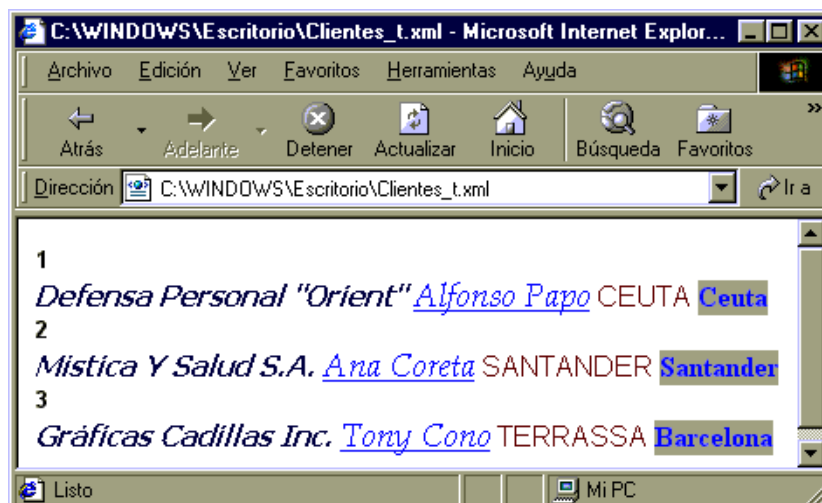


Figura 5. Salida en el Explorer del fichero anterior

Una de las características más sugestivas de las CSS, es que posibilitan al diseñador de sitios Web la creación de Hojas de Estilo que respondan a un mismo patrón, o *estilo corporativo*, al que basta con hacer referencia al principio de cada página web para poderlo utilizar, sin necesidad de repetir individualmente todos los elementos comunes. Es otra forma más de “sacar factor común”, como apuntábamos antes.

Presentación mediante Hojas de Estilo Extendidas (XSL)

Esta característica de las CSS, no dispone, sin embargo, de las posibilidades de procesamiento de los ficheros XSL. Con XSL disponemos de instrucciones para el tratamiento de conjuntos de elementos, o de elementos individuales, logrando formatear individualmente cada uno de ellos, o todos a la vez, y pudiendo establecer criterios de ordenación, condiciones de filtro, de navegación, etc.

Toda esta labor de selección, es, además, llevada a cabo por el analizador sintáctico del navegador, por lo que –una vez que los datos se han descargado– no hay que realizar más peticiones al servidor para

reorganizar la información, o para filtrarla o para remarcar un determinado aspecto. Como el lenguaje puede incluso evaluar sentencias JavaScript, cabe la posibilidad de implementar soluciones en las que el usuario seleccione el conjunto de registros a visualizar o el orden de salida, navegue a través de los elementos, elimine los que quiera, etc. Y todo en la máquina del cliente.

Nota: Respecto a los mecanismos asociados a lo que suele llamarse Tratamiento de datos, vea el lector el Capítulo 8 donde se comentan las soluciones disponibles para la gestión de información proveniente de servidores.

Plantillas XSL

XSL se basa en plantillas. El concepto de plantilla hay que entenderlo como una definición que permite la selección de un subconjunto de los registros existentes en un documento XML. La plantilla deberá incluir dos tipos de código: instrucciones XSL para seleccionar los elementos a mostrar, y etiquetas HTML (junto a estilos CSS) que –una vez efectuada la selección de registros- se apliquen sobre estos produciendo una salida formateada.

Ese subconjunto de los elementos puede abarcar a la totalidad de registros o limitarse exclusivamente a un único elemento o atributo. Para que tal selección sea posible, en XSL se han definido un conjunto de símbolos que permiten referenciar los elementos del árbol de datos en la forma que el usuario necesite. Concretamente, la siguiente tabla especifica cuáles son esos símbolos y su significado:

Patrones para la descripción de nodos en XSL

Símbolo	Descripción
/	Indica al procesador XSL que aplique una plantilla partiendo del nodo raíz. También se utiliza para indicar dependencia directa: Hijo de . Por ejemplo: ITEM/CATEGORÍA se refiere al elemento CATEGORÍA, que es hijo de ITEM.
//	Hace referencia a un nodo que es descendiente de otro (o del raíz) a cualquier nivel de profundidad. Por ejemplo, //CATEGORÍA indica que CATEGORÍA es descendiente de raíz a un cierto nivel de profundidad.
.	Se refiere al nodo actual (el que se está procesando en ese momento)
*	Hace referencia a cualquiera de los contenidos de un nodo
[]	Hace referencia a un elemento por su nombre y permite efectuar selecciones (Ver ejemplos más abajo). Permite funciones de posición, como end().
@	Hace referencia al contenido de un atributo

Tabla 9. Patrones para la descripción de nodos en XSL

Veamos algunos ejemplos de su uso para aclarar los significados, teniendo como modelo la Tabla 9.

/ se refiere al nodo raíz.

/Cliente se refiere a un nodo Cliente descendiente del nodo raíz.

// se refiere al nodo raíz y a todos los que dependen de él.

//Cliente se refiere a un nodo Cliente que es descendiente a algún nivel del nodo raíz.

//* se refiere a todos los nodos por debajo del nodo raíz.

./ todos los nodos descendientes del nodo actual.

./IdCliente todos los nodos IdCliente descendientes del nodo actual.

Cliente/* todos los hijos de Cliente.

Cliente//* todos los descendientes (a cualquier nivel) de Cliente.

Cliente/IdCliente se refiere al elemento IdCliente que es descendiente a algún nivel del nodo Cliente.

Cliente/@GRAFICO el atributo GRAFICO de un nodo Cliente.

Cliente/*@ se refiere a todos los atributos de un nodo Cliente.

Cliente[0] se refiere al primer elemento Cliente de la lista.

Cliente[end()] se refiere al último elemento Cliente de la lista.

Cliente[0]/IdCliente se refiere al elemento IdCliente del primer Cliente de la lista

Cliente[IdCliente] se refiere a aquellos elementos Cliente cuyo elemento IdCliente no esté vacío.

De la misma forma, éste mecanismo permite la selección de registros mediante expresiones de comparación. Para ello, entre corchetes, y respetando la sintaxis, se pueden indicar expresiones de comparación que servirán de filtro para aquellos registros que satisfagan la condición.

En este punto podemos toparnos con un inconveniente derivado de la sintaxis: ciertos símbolos están reservados para el lenguaje XML (>, <, “, etc.), por lo que el estándar propone operadores alternativos a utilizar en las operaciones de comparación sustituyendo a los tradicionales operadores relacionales. Dentro de la Tabla 10 enumeramos dicha lista (téngase en cuenta que algunos símbolos no generan problemas, como el signo =):

Operador	Equivalente XSL
<	\$lt\$
<=	\$le\$
>	\$gt\$
>=	\$ge\$
!=	\$ne\$
=	\$eq\$

Tabla 10. Operadores

Tabla de equivalencia XSL para los operadores relacionales

Según esto, podemos establecer operaciones de selección mediante expresiones como las siguientes:

Cliente[IdCliente \$eq\$ 1] selecciona el Cliente con el IdCliente = 1.

Cliente[IdCliente \$gt\$ 10] selecciona los Clientes cuyos IdCliente sean mayores que 10.

Cliente[Ciudad \$ge\$ 'Madrid'] selecciona los Clientes cuya Ciudad sea mayor o igual que "Madrid" (entendiendo >= según el código ASCII correspondiente).

Cliente[Ciudad = 'Madrid']/CODIGO selecciona los IdCliente de todos los Clientes de Madrid (hemos usado el signo = dado que no es incompatible con la sintaxis).

Para los atributos, la idea es la misma: si dispusiésemos de un atributo GRAFICO perteneciente a Cliente,

Cliente[@GRAFICO='Graficos/Imagen1.gif']

seleccionaría todos los Clientes cuyo atributo GRAFICO fuese 'Graficos/Imagen1.gif'.

Estas consideraciones de selección nos llevan inevitablemente al concepto de selección múltiple, tal y como podemos hacer en las cláusulas WHERE de la sintaxis SQL. En SQL y en otros lenguajes, cuando queremos que una expresión incluya más de una condición, debemos utilizar los operadores lógicos (AND, OR, NOT). También aquí muchos intérpretes admiten la sintaxis tradicional, pero además se han definido equivalentes de estos operadores para XSL, siguiendo la sintaxis coherentemente y utilizando, por tanto, los nombres de los operadores escritos en minúsculas y precedidos y sucedidos por el símbolo del \$. Así, AND se convierte en \$and\$, OR en \$or\$ y NOT en \$not\$.

Esto nos permite establecer criterios de selección complejos evitando ambigüedades, como por ejemplo:

Cliente[IdCliente \$ge\$ 2 \$and\$ Ciudad = 'Barcelona'] donde seleccionamos todos los Clientes cuyo IdCliente sea mayor o igual que 2 y cuya Ciudad sea Barcelona.

Obviamente, podemos realizar todo tipo de combinaciones posibles dentro de la lógica de los datos, siempre que respetemos la sintaxis. En caso de no escribir EXACTAMENTE la condición de filtro, XSL ignorará los elementos pudiéndose producir una salida vacía. Hay que tener en cuenta que, *en caso de que más de una plantilla fuera aplicable al mismo elemento, se seleccionará aquella que signifique una condición más restrictiva.*

Nota: Algunos intérpretes, como MSXML de Microsoft, permiten la inclusión de elementos HTML en las plantillas sin haber declarado las etiquetas básicas del lenguaje HTML (<HTML><BODY> y sus etiquetas de cierre correspondientes), sin embargo no tenemos garantía de que tal característica funcione para otros intérpretes, por tanto, lo recomendable es incluirlas en todos los documentos de presentación que utilicemos.

Veamos el equivalente al fichero CSS anterior, en versión XSL, pero con un factor añadido: el criterio de ordenación, que será ahora alfabético por el elemento Empresa (véase en la Figura 6).

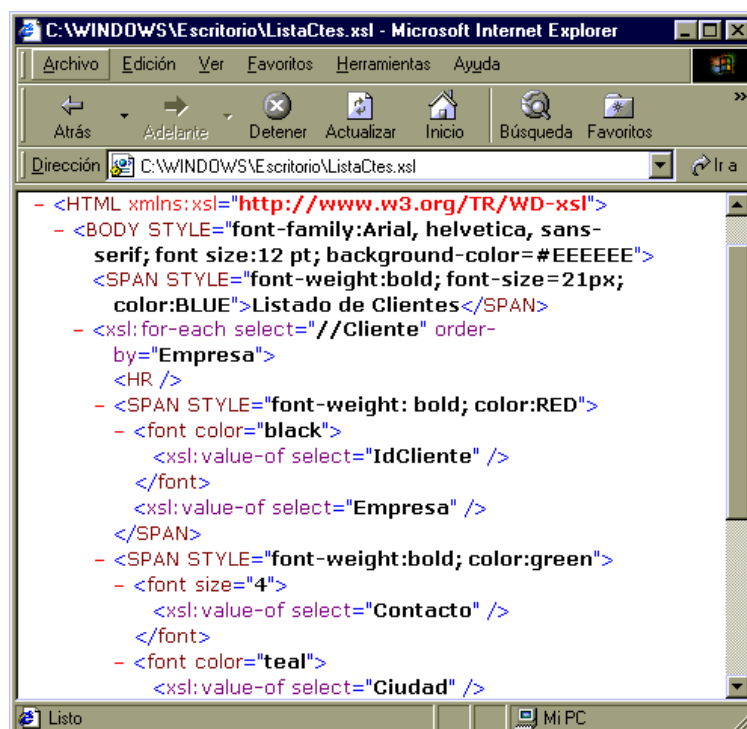


Figura 6. Modelo de fichero XSL tal y como es mostrado por el navegador Explorer 5.0

En este fichero de definición, apreciará el lector la presencia de un bucle `<xsl:for-each...>` que permite aplicar la definición interna del bucle a todo el conjunto de registros, sean los que sean, así como el atributo `order-by` que nos permite definir criterios de ordenación. La referencia a `Cliente` se hace mediante `select="//Cliente"`, que busca en la jerarquía de datos un elemento de ese nombre descendiente del nodo raíz a cualquier nivel, y aplica los estilos que se establezcan a cada elemento de ese nombre.

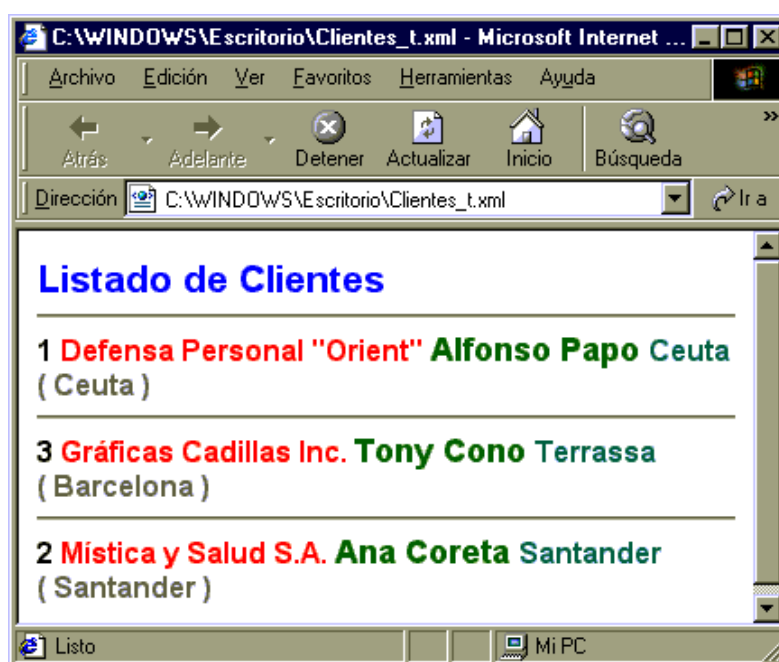


Figura 7. Salida en el navegador de los datos de ejemplo aplicando el modelo definido en el fichero XSL anterior

Los elementos `<xsl:value-of...>` se usan, junto a su atributo `select`, para establecer el elemento que desea visualizarse. Como vamos a ver, existe más de una instrucción XSL que se considera de aplicación global (o sea, que puede afectar a un conjunto de registros). La Figura 7 muestra la salida en el navegador Explorer, esta, no sólo permite el formato de elementos individuales, sino también, cómo se han reordenado los registros en orden creciente según las ventas.

El conjunto de instrucciones XSL

La Tabla 11 es una lista de instrucciones de procesamiento (las más utilizadas), junto a una breve descripción de su uso. Todas comienzan por el prefijo XSL, que hace las veces de espacio de nombres, seguido de una instrucción XSL.

Instrucción XSL	Descripción
<code>xsl:apply-templates</code>	Indica al procesador XSL que aplique una plantilla basado en el tipo y contexto de cada nodo seleccionado.
<code>xsl:attribute</code>	Crea un nodo atributo y lo adjunta a un elemento de salida.
<code>xsl:choose</code>	Junto a los elementos <code>xsl:otherwise</code> y <code>xsl:when</code> suministra un mecanismo de selección múltiple, al estilo de un Select Case de Visual Basic
<code>xsl:comment</code>	Genera un comentario en la salida
<code>xsl:copy</code>	Copia el nodo actual del origen a la salida
<code>xsl:element</code>	Crea un elemento con el nombre especificado
<code>xsl:eval</code>	Evalúa una expresión de script para generar una cadena de texto
<code>xsl:for-each</code>	Aplica una plantilla de forma repetida a un conjunto de nodos
<code>xsl:if</code>	Permite fragmentos de evaluación condicional simple
<code>xsl:otherwise</code>	Junto a los elementos <code>xsl:choose</code> y <code>xsl:when</code> suministra un mecanismo de selección múltiple, al estilo de un Select Case de Visual Basic
<code>xsl:pi</code>	Genera una instrucción de proceso en la salida
<code>xsl:script</code>	Define variables globales y funciones para extensiones de script
<code>xsl:stylesheet</code>	Declaración de tipo de documento para una hoja de estilo, conteniendo elementos <code>xsl:template</code> y <code>xsl:script</code>
<code>xsl:template</code>	Define una plantilla de salida para los nodos de un tipo y contexto en particular
<code>xsl:value-of</code>	Inserta el valor del nodo seleccionado como texto.
<code>xsl:when</code>	Junto a los elementos <code>xsl:choose</code> y <code>xsl:otherwise</code> suministra un mecanismo de selección múltiple, al estilo de un Select Case de Visual Basic

Tabla 11. Instrucciones de procesamiento.

Resultan especialmente potentes las instrucciones que permiten el tratamiento individual de campos, dependiendo del valor que contengan. Por ejemplo, la instrucción `<xsl:if...>` capacita la presentación de los datos en función del contenido, al igual que su homónima, `<xsl:when...>` que se utiliza junto a `<xsl:choose...>` y `<xsl:otherwise...>` para crear estructuras similares a la Select/Case de Visual Basic.

El Código Fuente 38 se muestra las instrucciones para el tratamiento de datos numéricos, consiguiendo con ello que los valores del campo Ciudad se mostraran en colores distintos según un patrón establecido.

```
<xsl:choose>
  <xsl:when test="Ciudad[. $eq$ 'Ceuta']">
    <font color="teal">
      <xsl:value-of select="Ciudad" />
    </font>
  </xsl:when>
  <xsl:when test="Ciudad[. $eq$ 'Santander']">
    <font color="red">
      <xsl:value-of select="Ciudad" />
    </font>
  </xsl:when>
  <xsl:otherwise>
    <font color="black">
      <xsl:value-of select="Ciudad" />
    </font>
  </xsl:otherwise>
</xsl:choose>
```

Código Fuente 38. Modificación de la salida del campo Ciudad en función del contenido

Obtenemos así una salida condicionada, tal y como se muestra en la Figura 8.

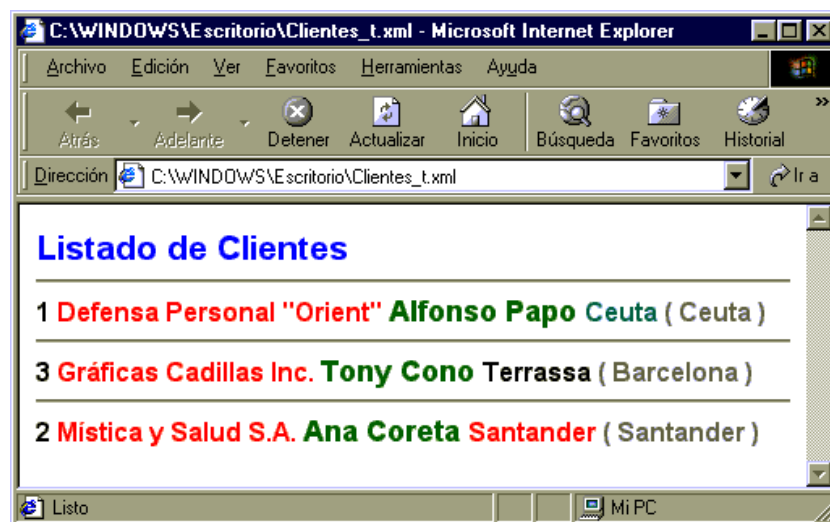


Figura 8

Para terminar este recorrido por las características de XSL, diremos que XSL también permite una integración total con los lenguajes de script (JavaScript, VBScript), a través de su atributo `language` y

de la instrucción que evalúa una expresión válida en el lenguaje especificado, incluyendo la interpretación de funciones, como se demuestra en el Código Fuente 39.

```
<xsl:stylesheet language="VBScript" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <xsl:eval>Celsius(32)</xsl:eval>
  </xsl:template>

  <xsl:script language="VBScript">
    Function Celsius(Grados)
      Celsius = (Grados - 32) * 5 / 9
    End Function
  </xsl:script>
</xsl:stylesheet>
```

Código Fuente 39. Inclusión de una referencia a un lenguaje de Script en un fichero XSL

Como es fácil de interpretar, la sentencia `<xsl-eval...>` buscará una función de nombre Celsius y le pasará como parámetro el valor 32, la función se ejecutará y el valor de retorno será interpretado como dato a mostrar en la página.

Ejercicios

1. ¿Qué función desempeñan los ficheros XSL?
2. ¿En qué se diferencian de las Hojas de Estilo?
3. ¿Dispone XSL de estructuras de control?
4. ¿Dispone XSL de mecanismos de evaluación de expresiones? ¿Cuáles?
5. ¿Cómo evaluaríamos la expresión $3+(4-6) * 2$?
6. ¿Cómo podemos variar dinámicamente una lista de datos de salida?
7. ¿Cómo se controla la apariencia dependiendo del contenido?
8. ¿Cómo se invoca un fichero XSL externo?
9. ¿Cómo se integra XSL con lenguajes de Script? Pon un ejemplo de su funcionamiento.

El modelo de objetos de documento(DOM)

DOM no es un modelo específico de XML sino precisamente lo que *convierte* al HTML estático en dinámico, y que podemos entender como la forma en la que los exploradores interpretan una página que por su naturaleza es estática (o desprovista de *comportamientos programables*), transformando sus elementos en objetos, que, como tales, poseen propiedades, métodos y eventos, y que por lo tanto, se convierten en *entidades programables*.

A los lenguajes de programación que nos permiten programar estos objetos DHTML, se les denomina Lenguajes de Script (principalmente, Javascript y VBScript, basados, respectivamente, en sus *hermanos mayores*, Java y Visual Basic). Un objeto DHTML se programa asignándole un identificador (un valor para su atributo ID, lo que lo convierte en objeto programable) y estableciendo una acción, escrita en uno de estos lenguajes, y asociada con uno cualquiera de los eventos de que el objeto disponga.

El esquema de la Figura 9 ilustra la forma en la que viaja y se transforma la información de estática a dinámica.



Figura 9. Esquema de envío y transformación de una página web mediante DOM

Cuando un usuario solicita una página web (por ejemplo, <http://www.eidos.es>), el servidor web busca dicha página, la envía al cliente y allí sufre un proceso de transformación: primero se lee todo el contenido; a continuación, se construyen tantos objetos en la memoria como elementos de la página HTML tengan un identificador (ID), y finalmente, se da un formato gráfico de salida al documento, al tiempo que el motor del navegador *permanece a la escucha* de los eventos que el usuario genere al navegar por la página. Cuando se produce uno, (como pasar el cursor por encima de un ítem de menú, o de un gráfico), el *parser* llama al intérprete del lenguaje de script que corresponda, y la acción se ejecuta. Esa es la forma en que los menús cambian de color o de tamaño cuando navegamos por ellos, y también la forma en la que se producen un sinnúmero de efectos especiales que estamos ya acostumbrados a ver en las páginas web.

La llegada de XML no supone un cambio de esa filosofía, al contrario. Más bien supone una expansión de ese concepto para que en lugar de tener que hacer un tratamiento individualizado de cada objeto, podamos trabajar con grupos de objetos (por ejemplo registros) dotándoles de un comportamiento similar al que puedan tener dentro de una interfaz de usuario tradicional, al estilo de *Visual Basic* o *Delphi*: esto es, para que podamos realizar búsquedas o reordenaciones (lo veíamos en el capítulo anterior, sobre XSL) o cualquier otra operación que pudiéramos realizar sobre una interfaz de usuario tradicional dentro de esos grupos de objetos.

Una vez más, es labor del navegador (o de las librerías que realizan la interpretación o rendering) el construir objetos dinámicos a partir de lo que sólo es un documento, evitando así, el envío de componentes a través de la web. En la Figura 10 mostramos la jerarquía de objetos de DOM, en su versión inicial (que algunos autores también denominan DHTML Object Model), o sea antes de la aparición de XML.

Observamos que la jerarquía se basa sobre todo en dos objetos fundamentales: el objeto window, que refleja la estructura y propiedades del navegador, y el objeto document (uno sólo, y no una colección, ya que se trata de una interfaz SDI), que contiene todo lo referente a la página web que se visualiza en un momento dado. Como podemos ver, algunos objetos pasan a pertenecer a colecciones concretas, como la colección de imágenes (images) o la colección de hojas de estilo (styleSheets). Otros, por su importancia se transforman en objetos individuales, como body o element, y además existe una colección donde van parar todos los objetos programables: all.

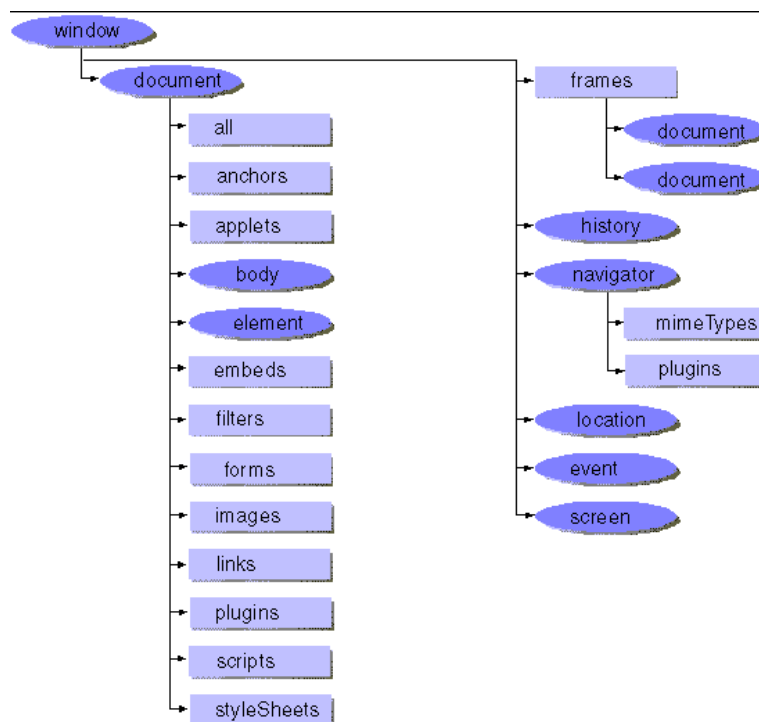


Figura 10. Modelo de objetos DHTML (fuente:MSDN)

El Modelo de objetos de documento XML

En el modelo de objetos con soporte XML, cuando se carga un documento XML existe un objeto especial, denominado Document (implementado en la interfaz como XMLDOMDocument), del cual dependen todos los objetos de la jerarquía, y que podemos interpretar como el nodo raíz, o elemento principal. De él, penden objetos Node (o concretamente, XMLDOMNode), y objetos NamedNodeMap (XMLDOMNamedNodeMap), que permiten el acceso a los valores de los atributos, además de incluir soporte para los espacios de nombres (namespaces). El objeto NodeList (XMLDOMNodeList), por su parte, permite acceder a los nodos de forma global, al tratarse de un objeto *collection* que contiene la lista de los nodos disponibles. Haciendo una abstracción de los objetos más importantes de cara al programador (la lista completa es larguísima), podemos hacernos una idea de la situación mediante el diagrama de la Figura 11.

El diagrama muestra como además de los objetos estándares, el soporte de IE5 añade ciertas funcionalidades no aprobadas todavía por W3C, para facilitar el manejo de los documentos XML. Como ya sabemos, su utilización dependerá del grado de seguridad que tengamos sobre el navegador o intérprete a utilizar en cada caso en particular.

En detalle, el objeto ParseError (IDOMParseError) informa acerca de los problemas que pudiera encontrar el propio intérprete a la hora de transformar el documento XML, mientras que el objeto XMLHttpRequest (IXMLHttpRequest), permite la gestión de un protocolo de comunicaciones con servidores HTTP en la parte del cliente. Respecto al objeto XMLHttpRequest (IXMLHttpRequest), se utiliza para permitir la manipulación de ficheros de presentación escritos en XSL, y asociados con el documento XML.

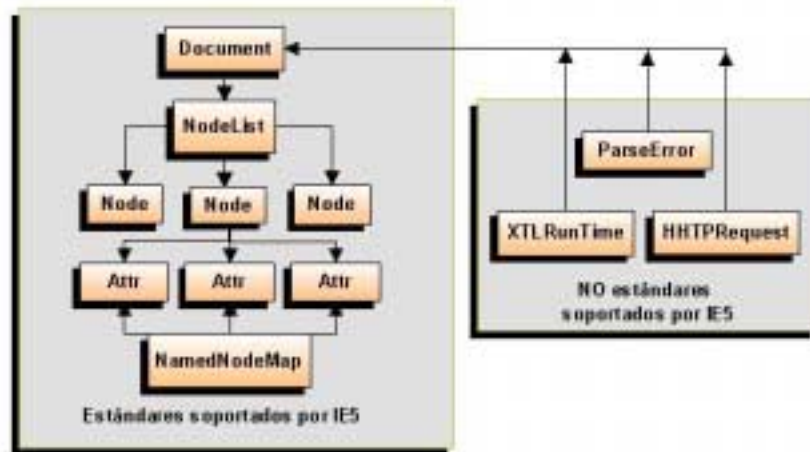


Figura 11. Diagrama básico del modelo de objetos XML

Quizá se pregunte el lector qué pasa con el resto de elementos que hemos estudiado como parte del lenguaje XML: como es lógico también están presentes, y, en su mayoría, son manejables a partir de los objetos que hemos listado anteriormente. Veremos algunos de ellos en los ejemplos que siguen a continuación. El lector que necesite información adicional sobre toda la jerarquía de objetos del modelo XML DOM, podrá encontrarla en su totalidad en la dirección Internet: <http://www.w3.org/TR/REC-DOM-Level-1/>, o en cualquiera de las direcciones de desarrolladores XML recomendadas al principio de éste curso.

Lo más importante antes de trabajar con XML DOM es tener presente que todo el contenido de un documento se ve desde la jerarquía como un conjunto de nodos. ¿Por qué nodos en lugar de elementos? Porque una de las diferencias principales con la jerarquía DHTML es que allí encontrábamos colecciones que debían existir *siempre*, independientemente de que contuviesen o no elementos. Por ejemplo, siempre encontramos una colección `images`, aunque la página no contenga una sola imagen. En XML, la situación es más flexible: no existen una serie de colecciones predefinidas de objetos hasta que no ha concluido el proceso de transformación. La única cosa que sabemos con seguridad, es que habrá un objeto `element`, que se corresponderá con el raíz. Todo lo demás dependerá del contenido del documento.

Por eso, lo mejor es imaginarse el contenido como un estructura de árbol, y cada ítem como un nodo genérico. Algunos nodos equivaldrán a las *hojas*, que no contendrán más sub-elementos. Otros serán equivalentes a los troncos y su misión será contener una serie de nodos hijos (child nodes). La complejidad inicial surge del hecho de que la transformación considera como nodos individuales tanto los elementos contenedores (las marcas mismas) como los contenidos (lo que hay entre ellas). Veamos un ejemplo: el elemento categoría de nuestro ejemplo de la lista de libros se expresa en el documento como aparece en el Código Fuente 40.

```
<Ciudad>Sevilla</Ciudad>
```

Código fuente 40. Nodo estándar hijo de <Cliente>

Bien, pues, para el parser, existirá un nodo `Ciudad` que será de tipo `element`, pero la cadena "Sevilla", que está contenida en él, también será un nodo, pero de tipo `Text`. La propiedad `value`, del nodo `Ciudad` valdrá `null`, pero dispondrá de un nodo hijo cuya propiedad `value` será "Sevilla". Veremos

esto más detalle a través de los ejemplos. Así pues, los nodos nos dan mediante sus propiedades y métodos, la posibilidad de averiguar todo lo referente a cada elemento de un documento. Un nodo sólo podrá tener un nodo padre, pero podrá contener una colección de nodos hijos (con la sola excepción del nodo raíz, que no tiene nodo padre y sólo puede tener un nodo hijo, el principal de la jerarquía, que en nuestro ejemplo hemos llamado Clientes, por tanto el nodo raíz es como si hiciera referencia al documento mismo). Por su parte, los elementos pueden tener atributos, y estos atributos son -a su vez- objetos nodo, sólo que de un subtipo diferente al del resto de nodos.

Los nodos

Cada nodo tiene una propiedad `childNodes`, que hace referencia a sus nodos hijos, y una propiedad `attributes`. Si el nodo contiene algún atributo, la propiedad devuelve una referencia al objeto `NamedNodeMap`, que contiene los objetos nodo de tipo `attribute`, en caso contrario, devuelve `null`. En el primer caso, si el nodo contiene nodos hijos, la propiedad `childNodes`, devuelve una referencia al objeto `NodeList` que contiene los nodos hijos. En caso contrario, también devuelve `null`.

Cada nodo, por su parte, dispone de los métodos necesarios para acceder a aquellos nodos que se encuentran vinculados con él mediante relaciones jerárquicas. En el caso de utilizar IE5, dispondremos, además, de algunas propiedades específicas, para consultar valores como el namespace, el prefijo (prefix), o la definición de un elemento.

En otro orden de cosas, los nodos no son algo preestablecido y fijo. El usuario puede, dinámicamente crear nuevos nodos y cambiar características de los ya existentes. Para ello, cada nodo dispone de un conjunto de métodos, algunos estándares y otros específicos de cada navegador, que permiten la manipulación programática de elementos y contenidos.

Objetos de alto nivel de la jerarquía DOM

Hemos comentado algunas peculiaridades de los objetos básicos, pero vamos ahora a revisar los objetos de más alto nivel en la jerarquía, que añaden características extendidas al modelo DOM. Como ya se ha citado, un documento XML se representa como una estructura que comienza con un objeto (nodo) `Document`. A su vez, éste nodo puede tener su colección de nodos hijos (`childNodes`) y su colección de atributos (`attributes`), además de otros nodos que representan otros aspectos considerados de los documentos: por ejemplo, el objeto `DocumentType`, hace referencia al DTD de un documento, que tiene su propia colección de entidades (`entities`), notaciones (`notations`), etc.

El elemento primario que desciende de `Document`, es el nodo principal, dentro de cuyas etiquetas se encuentran todos los demás. Un diagrama más completo que el de la Figura 11 podría ser, el esquema que se muestra en la Figura 12.

Hemos visto hasta aquí algunos de los elementos principales de la jerarquía DOM, tal y su estructura y descripción funcional básica. Pasemos ahora a la manipulación de los datos, desde un doble enfoque: el manejo de DOM desde el propio navegador, y la utilización -cada vez más importante- de los documentos XML como formato de intercambio de datos, para ser posteriormente leídos con una herramienta de desarrollo, como Visual Basic.

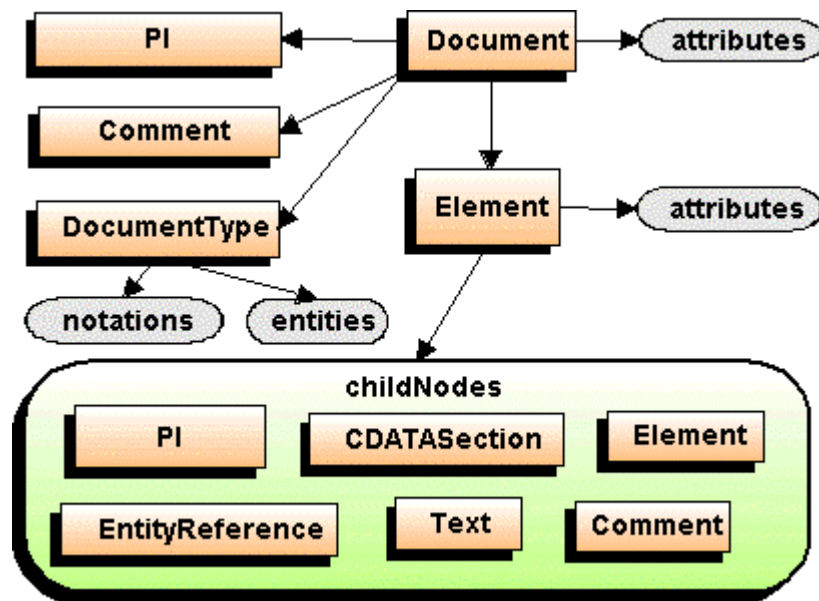


Figura 12. Diagrama del modelo XML con objetos de alto nivel

Ejercicios

1. ¿En qué se basa el modelo de objetos de documento?
2. ¿Cuáles son las diferencias principales entre DOM estándar y XML DOM?
3. En el modelo DOM estándar, ¿se tiene acceso a los formularios de una página HTML?
4. ¿Qué lenguajes podemos usar para la creación de documentos dinámicos?
5. ¿Cuál es el de más amplio reconocimiento?
6. ¿Cuál es el objeto principal de la jerarquía XML DOM?
7. ¿Cuál es el objeto más ampliamente utilizado cuando se trabaja con XML DOM?
8. Una de las diferencias importantes entre DOM y XML DOM es que en el segundo no existen colecciones “obligatorias”. ¿Qué queremos decir con esto?
9. ¿Existe alguna entidad de un DTD que no sea accesible a través de XML DOM?

XML DOM en la práctica

Desde el Explorer, podemos instanciar un objeto Parser utilizando un lenguaje de script, y posteriormente, cargar el documento y acceder a los nodos y a sus valores a partir de los métodos que el objeto proporciona.

Veamos un ejemplo de utilización en el Código Fuente 41.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE = VBScript>
Dim xmlDoc
Dim NodoActual
Dim NodoHijo
Set xmlDoc = CreateObject("microsoft.xmlDOM")
xmlDoc.async = False
xmlDoc.load("C:/WINDOWS/Escritorio/Clientes_t.xml")
Set NodoActual = xmlDoc.documentElement.childNodes.item(0)
MsgBox NodoActual.text
MsgBox NodoActual.xml
Set Hijo = NodoActual.childNodes.item(3)
Msgbox Hijo.text
</SCRIPT>
</HEAD>
<BODY>
Primera prueba de uso del XML DOM
</BODY>
</HTML>
```

Código Fuente 41. Código HTML con una llamada al Parser MSXML

Aunque la página web no contiene ningún texto, la presencia de un código VBScript sin estar incluido en ninguna función, hará que el navegador ejecute el código antes de interpretar el resto de la página. Primero, creamos un objeto **xmlDoc**, que hace referencia al propio Parser. Como estamos en VBScript, utilizamos la sentencia *CreateObject*, seguido del nombre del objeto, y le indicamos antes de abrir ningún documento que preferimos que no continúe la ejecución hasta que se haya cargado completamente (propiedad **async** = false).

El siguiente paso es cargar el documento XML, lo que hacemos mediante el método **load** del objeto Parser que hemos instanciado (**xmlDoc**). A partir de éste momento, ya es posible utilizar los objetos disponibles en la jerarquía, para tener acceso a los diferentes nodos contenidos en el documento. Como una primera aproximación, usamos la propiedad **documentElement**, que hace referencia al primer elemento, y dentro de la colección de nodos (**childNodes**), seleccionamos el primero (**item(0)**). Una vez hecho esto, podemos acceder a los contenidos, o a todo el texto en formato XML usando las propiedades homónimas **text** y **xml**.

Las salidas, son sendas cajas de texto con la información en los dos formatos. En el primer MsgBox, obtendríamos la Figura 13, esto es, la información devuelta por la propiedad **text**, resume en una sola línea todos los datos del primer elemento Cliente, o sea, los valores de los elementos CODIGO, CATEGORIA, FECHA_SALIDA, TITULO y VENTAS.



Figura 13. Datos del primer nodo del fichero de Clientes

En el segundo Msgbox (véase Figura 14), sin embargo, la información es completa, recogiendo el *contenido XML* del primer nodo.

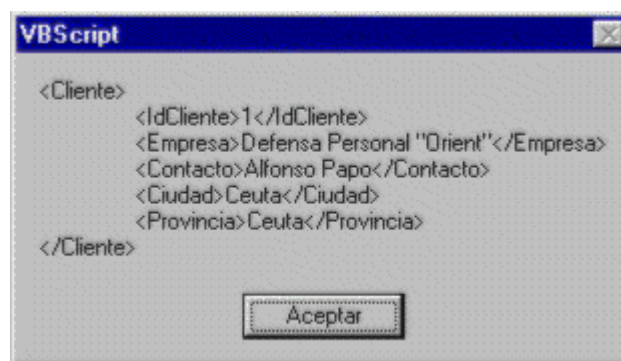


Figura 14. Primer nodo más etiquetas XML

y podemos acceder a los nodos individuales, siguiendo la misma filosofía, leyendo los hijos del nodo seleccionado; Las líneas siguientes permiten leer el contenido individual del elemento Ciudad (cuarto nodo hijo de Cliente), obteniendo dicho valor individual (Figura 15).



Figura 15. Valor del Campo Ciudad del 1º Nodo

y así podríamos continuar hasta el último elemento de la jerarquía. Paralelamente, existen otras opciones que nos permiten acceder al valor de un nodo sin necesidad de conocer su número de orden. Por ejemplo, el método **selectSingleNode(<"NOMBRE DEL NODO">)**, permite obtener el mismo resultado que en el caso anterior (visualizar el título del primer libro), con el Código Fuente 42.

```
Set currNode = xmlDoc.documentElement.selectSingleNode("Cliente/Empresa")
MsgBox currNode.text
```

Código Fuente 42. Método de acceso alternativo a un nodo mediante su nombre

Es importante recalcar que el que realiza la acción de abrir el documento y exponer la jerarquía de objetos programable es MSXML, y por tanto podemos pensar en un modo alternativo de hacer exactamente lo mismo, pero utilizando JavaScript en lugar de VBScript, consiguiendo una total independencia del navegador (si bien, MSXML debe estar presente y registrado en la máquina desde la que se abra el documento XML). El Código Fuente 43, ejecuta exactamente las mismas salidas, pero utiliza JavaScript en la implementación.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = false;
    xmlDoc.load("file://C:/WINDOWS/Escritorio/Clientes_t.xml");
    var currNode = xmlDoc.documentElement.childNodes.item(0);
    alert(currNode.text);
    alert(currNode.xml);
    var NodoHijo = currNode.childNodes.item(3);
    alert(NodoHijo.text);
    var NodoHijo = xmlDoc.documentElement.selectSingleNode("Cliente/Ciudad");
    alert(NodoHijo.text);
</SCRIPT>
</HEAD>
<BODY> </BODY>
</HTML>
```

Código Fuente 43. Código equivalente al anterior utilizando JavaScript

Un ejemplo completo

Si nos planteamos un ejemplo completo, podemos ver más en detalle las posibilidades que esta técnica ofrece, en particular implementar una navegación al estilo de los formularios de datos que utilizamos habitualmente en las herramientas de desarrollo.

No se precisan muchos cambios para ver este mecanismo en funcionamiento. Únicamente, debemos crear una página Web en la que un formulario contenga objetos que permitan visualizar la información leída, y –eventualmente– un sistema de navegación que posibilite el desplazamiento por los distintos nodos. Dado que tenemos 5 elementos por nodo base (IdCliente, Empresa, Contacto, Ciudad y Provincia), insertaremos en nuestra página web 5 cajas de texto para mostrar la información (podemos usar cualquier otro mecanismo válido incluso elementos que no pertenezcan a formularios HTML como etiquetas <DIV>).

Además, y para permitir la navegación añadiremos dos botones, -Siguiente y Anterior- a los que asignaremos un código adecuado que permita leer los nodos siguiente y anterior respectivamente. De forma parecida a lo que hacemos en otros lenguajes, rellenaremos las cajas mediante una rutina que recorra los elementos de un nodo base y vaya asignando a cada caja de texto el valor de la propiedad **text** obtenido. En resumen, nuestra página web resultante se compondría del Código Fuente 44.

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso de XML DOM bajo Internet Explorer 5.0</TITLE>
<SCRIPT LANGUAGE=javascript>
    //INSTANCIAMOS XML DOM en forma global
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = false;
    //Abrimos el documento (también global)
    xmlDoc.load("file:///D:/WINNT/Profiles/mposadas/Escritorio/Clientes_t.xml");
    //Y obtenemos el primer nodo en la variable global currNode
    var currNode = xmlDoc.documentElement.childNodes.item(0);

    function Anterior() {          //Se desplaza al nodo anterior
        currNode = currNode.previousSibling;
        Rellenar();
    }
    function Siguiente() {        //Se desplaza al nodo siguiente
        currNode = currNode.nextSibling;
        Rellenar();
    }
    function Rellenar() {
        var NodoHijo = currNode.childNodes.item(0);
        text1.innerText = NodoHijo.text;
        var NodoHijo = currNode.childNodes.item(1);
        text2.innerText = NodoHijo.text;
        var NodoHijo = currNode.childNodes.item(2);
        text3.innerText = NodoHijo.text;
        var NodoHijo = currNode.childNodes.item(3);
        text4.innerText = NodoHijo.text;
        var NodoHijo = currNode.childNodes.item(4);
        text5.innerText = NodoHijo.text; }
</SCRIPT>
</HEAD>
<BODY LANGUAGE=javascript onload="return Rellenar()">
```

Código Fuente 44. 1ª Parte del código del ejemplo completo de navegación de nodos de un documento XML (hasta la etiqueta <BODY>)

Hemos preferido sacrificar la elegancia del código en aras de la claridad, por lo que no utilizamos un array de controles, y la función `Rellenar()` repite el mismo código con una mínima variación para cada uno de los valores de los cinco elementos.

Esta es la parte importante respecto a la programación. El método `Rellenar()`, como se ha dicho, utiliza cinco llamadas casi idénticas para rellenar las cajas de texto con el dato obtenido a partir del nodo, asignando la propiedad **innerText**.

Los procesos de instanciación del Parser, apertura del documento y obtención del primer nodo utilizable, se han situado fuera de cualquier función para que las variables sean globales en todo el `<SCRIPT>`. Finalmente, para que al abrir el documento se muestre el contenido del primer nodo, utilizamos el evento **onload()** del objeto **window**, haciendo que devuelva el resultado de llamar a la función **Rellenar()** (téngase en cuenta que *antes de ejecutar* `Rellenar()` se ejecutan las instrucciones situadas fuera de las funciones).

El Código Fuente 45, muestra la continuación del código HTML correspondiente, que construye las cajas de texto y los dos botones Anterior y Siguiente, usando un mínimo formato de salida.

```
<P><FONT size=5>Listado de Libros</FONT><HR size=2>
<table border="1" width="36%">
  <tr>
    <td width="39%" bgcolor="#FFFF00">Código </td>
    <td width="61%" bgcolor="#00FFFF"><font color="#800000">
<INPUT id=text1 name=text1></font></td>
  </tr>
  <tr>
    <td width="39%" bgcolor="#FFFF00">Categoría </td>
    <td width="61%" bgcolor="#00FFFF"><font color="#800000">
<INPUT id=text2 name=text2></font></td>
  </tr>
  <tr>
    <td width="39%" bgcolor="#FFFF00">Fecha de Salida</td>
    <td width="61%" bgcolor="#00FFFF"><font color="#800000">
<INPUT id=text3 name=text3 style="LEFT: 320px; TOP: 92px"></font></td>
  </tr>
  <tr>
    <td width="39%" bgcolor="#FFFF00">Título </td>
    <td width="61%" bgcolor="#00FFFF"><font color="#800000">
<INPUT id=text4 name=text4></font></td>
  </tr>
  <tr>
    <td width="39%" bgcolor="#FFFF00">Ventas </td>
    <td width="61%" bgcolor="#00FFFF"><font color="#800000">
<INPUT id=text5 name=text5 style="LEFT: 10px; TOP: 121px"></font></td>
  </tr>
</table>
<P>
<INPUT id=button1 language=javascript name=button1 onclick=Anterior() type=button
value=Anterior>
<INPUT id=button1 language=javascript name=button1 onclick=Siguiente() type=button
value=Siguiente></P>

</BODY>
</HTML>
```

Código Fuente 45. Código equivalente al anterior utilizando JavaScript

Mediante este mecanismo, obtendríamos una pantalla de salida en el navegador similar a la Figura 16.

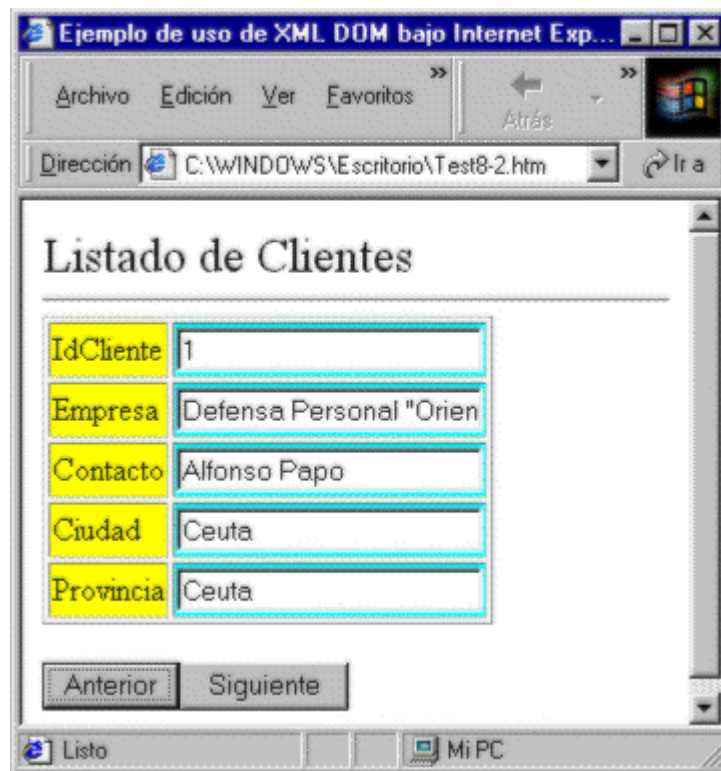


Figura 16. Salida el ejemplo final de uso de XML DOM en Internet Explorer 5

Donde el usuario podrá comprobar que existe una posibilidad de navegación, que podríamos extender a las búsquedas, filtros de registros, criterios de ordenación, etc., y todo ello sin necesidad de posteriores llamadas al servidor. Es la máquina cliente la que mantiene la información, la procesa y la presenta. Eventualmente, podríamos incluir un tratamiento de la presentación mediante un fichero XSL adecuado.

Utilización de XML DOM con herramientas de desarrollo

La ventaja de exponer el analizador sintáctico de XML como una librería de automatización es que puede ser utilizado desde otras herramientas, como Visual Basic, Delphi o incluso Microsoft Office. Todo se reduce a establecer una referencia a la librería desde la herramienta en cuestión, e implementar una interfaz de usuario desde la que poder acceder a los nodos y rellenar un control (o un documento, si estamos usando Office) con el contenido que hemos leído.

XML desde Visual Basic

Como es lógico, aunque la forma de trabajo sea la misma que hemos visto para el navegador hay aspectos que cambian al hacer el tratamiento de documentos desde una herramienta de desarrollo. Para empezar, será preciso utilizar un formulario para mostrar la información, e incluir algunos controles para mostrar los datos. Además, tendremos que hacer referencia a la librería XML DOM y crear algunos controles adicionales que suministren la funcionalidad que queremos suministrar a la interfaz.

Antes de comenzar, pues, (y dentro de un proyecto estándar de Visual Basic), realizamos una referencia a la librería “Microsoft XML 1.0”, que nos aparecerá en el apartado de Referencias disponibles. (Véase Figura 17)

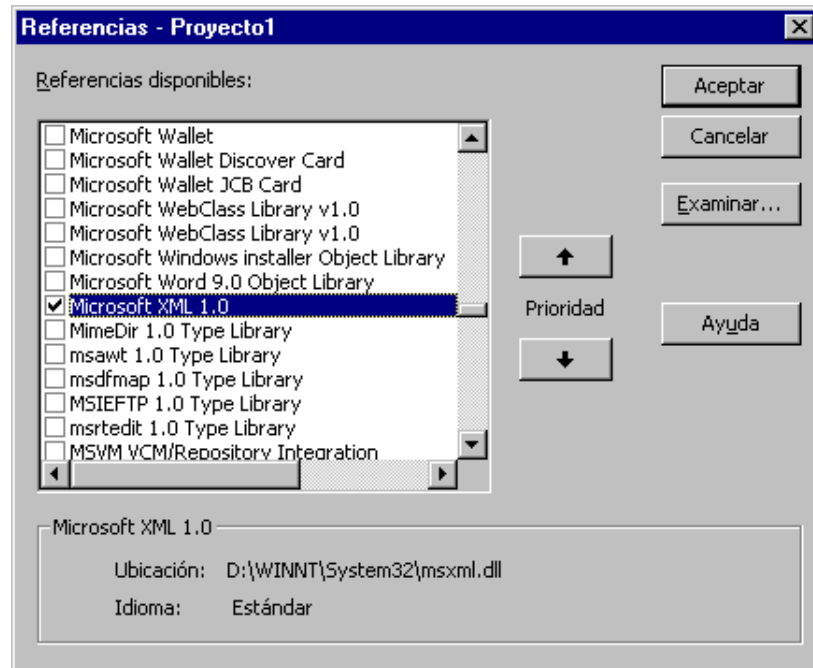


Figura 17. Referencia a XMLDOM desde V.Basic

Una vez realizados los requisitos previos, el primer paso sería declarar una variable de instancia para XML DOM y aprovechar el evento Load del formulario de datos para realizar la carga del documento XML en forma asíncrona y establecer el primer nodo de trabajo. Además, llevaremos el control del número de registros (nodos) leídos en el documento, a título informativo.

La variable `xmlDoc` hace referencia al propio documento, `colNodos` es la colección de todos los nodos disponibles en el documento, lo que se implementa en la librería MSXML mediante la interfaz **IXMLDOMNodeList**. La variable `Nodo`, a su vez servirá para el control de cada nodo individual, implementado mediante la interfaz **IXMLDOMNode**, y finalmente, `NumReg` servirá para mostrar el número de nodos, información que es aportada por la propiedad **length** del objeto `colNodos`.

Para mostrar la información leída, hemos optado en este ejemplo por la interfaz clásica, usando un array de cinco **TextBoxes** y cinco **Labels**, que son poblados en cada llamada al método *Rellenar()*. Esta función, recorre los nodos hijos de cada nodo principal, y asigna la propiedad *Text* a cada propiedad *Text* del array de **TextBoxes** creado para mostrar la información. Lo mismo sucede con la propiedad *Caption* de las etiquetas, sólo que esta vez usaremos la propiedad *baseName*, del nodo.

```
Dim xmlDoc As New MSXML.DOMDocument
Dim colNodos As MSXML.IXMLDOMNodeList
Dim Nodo As MSXML.IXMLDOMNode
Dim NumReg As Long

Private Sub Form_Load()

    'Establecemos la carga en modo asíncrono
```

```

xmlDoc.async = False
'Cargamos el documento
xmlDoc.Load "C:/WINDOWS/Escritorio/Clientes_t.xml"
'Inicializamos la colección de nodos
Set colNodos = xmlDoc.documentElement.childNodes
'Calculamos el número de registros
NumReg = colNodos.length
Frame1.Caption = Frame1.Caption & ": " & NumReg
'Obtenemos el primer nodo Hijo (del nodo raíz)
Set Nodo = colNodos.Item(0)
MsgBox Nodo.baseName
Rellenar

End Sub
Function Rellenar()

'Para controlar lecturas fuera de límite
On Error GoTo Errores
Dim i As Integer, NodoHijo As MSXML.IXMLDOMNode

For Each NodoHijo In Nodo.childNodes
    Etiqueta(i).Caption = NodoHijo.baseName
    txtNodo(i).Text = NodoHijo.Text
    i = i + 1
Next
Exit Function

Errores:
MsgBox "Error de lectura"

End Function

```

Código Fuente 46. Instanciación de un objeto XML DOM desde Visual Basic 6.0

Finalmente y con objeto de completar mínimamente la interfaz mediante una navegación básica, incluimos 4 botones de desplazamiento, a los que asignaremos código correspondiente a los métodos de movimiento por los nodos que exponen los objetos instanciados.

En concreto, las propiedades *nextSibling* y *previousSibling* permiten el desplazamiento al siguiente (y al anterior) de los nodos **hermanos** de aquel en el que nos encontramos, y el control del primero y el último se realiza mediante la variable NumReg, ya mencionada.

La codificación de esta parte final adoptaría la forma del Código Fuente 47 y ofrecería como salida, la Figura 18.

```

Private Sub cmdPrimero_Click()
    Set Nodo = colNodos.Item(0)      'Se desplaza al primer nodo
    Rellenar
End Sub
Private Sub cmdAnterior_Click()
    Set Nodo = Nodo.previousSibling  'Se desplaza al nodo anterior
    Rellenar
End Sub
Private Sub cmdSiguiente_Click()
    Set Nodo = Nodo.nextSibling      'Se desplaza al nodo siguiente
    Rellenar
End Sub
Private Sub cmdUltimo_Click()
    Set Nodo = colNodos.Item(colNodos.length - 1)      'Se desplaza al último nodo

```



```
Rellenar
End Sub
```

Código Fuente 47. Continuación del código anterior

Figura 18. Formulario VB procesando el 1º nodo del fichero de Clientes

Generación de ficheros XML a partir de recordsets de ADO

Por el momento, una de las características más utilizadas para construir ficheros XML es la utilización del método `Save` de un recordset de ADO. A partir de la versión 2.1, cada objeto Recordset puede almacenarse en disco usando dos formatos posibles: **adPersistDTG** (formato binario, propietario de Microsoft), y **adPersistXML**, que produce un fichero en texto plano, a partir del recordset que haya en memoria, descrito mediante un XML-Schema, que ADO genera a partir de los metadatos del *recordset* en cuestión. Para evitar ambigüedades, ADO genera en dicho Schema un conjunto de espacios de nombres únicos que evitan que pueda haber conflictos de interpretación.

A la vista del fichero de salida que genera ADO, el lector puede sorprenderse de la forma en que se encuentran almacenados los datos en el fichero: ADO genera un nodo raíz, llamado `<rs:data>`, (`rs` es un espacio de nombres definido previamente) y un conjunto de elementos `<z:row>` (donde `z` es otro espacio de nombres predefinido), y mete todos los datos del recordset como atributos de cada elemento `<z:row>`. De tal forma, que un registro (elemento) creado mediante este mecanismo, tiene el aspecto que aparece en el Código Fuente 48.

```
<z:row Codigo="0001" Categoria="XML Básico" Fecha_Salida="09-09-99"
Titulo="Introducción a XML" Ventas="500000"></z:row>
```

Código Fuente 48. Elemento fila (`z:row`) generado por ADO 2.1 a partir de un Recordset

De esta forma, ADO consigue algo importante: desprenderse de los problemas derivados del análisis sintáctico de los contenidos de cada elemento: en efecto, si incluimos los datos dentro de una etiqueta, probablemente hayamos especificado que su contenido es #PCDATA, dentro de la definición de su DTD. Eso quiere decir que cualquier carácter no ASCII que figure como contenido de un elemento, será analizado, y producirá un mensaje de error.

Por el contrario, incluyendo los datos en atributos (que normalmente se definen como CDATA), el *parser* no analizará los contenidos y dejara en manos de la aplicación que los lea la interpretación de los mismos.

A modo de prueba hemos construido una pequeña aplicación en Visual Basic, que obtiene un conjunto de resultados, y posteriormente los graba en formato XML. Como se pretende ver el grado de manipulación que puede obtenerse con herramientas de desarrollo, lo que hacemos es producir una transformación en el fichero original: primero eliminamos el Schema generado por ADO y lo sustituimos por un DTD apropiado.

Como es lógico, ese DTD será dependiente del conjunto de registros seleccionado, por lo que la aplicación deberá de analizar los nombres de los campos de dicho recordset, y crear el DTD consecuentemente, pero sabiendo cuál es la forma de construcción de una estructura de datos ADO. Finalmente, eliminamos los espacios de nombres, y grabamos el fichero en el nuevo formato, al objeto de que pueda visualizarse con ayuda de un fichero de presentación escrito en lenguaje XSL, al que podemos invocar desde la propia aplicación, mediante un control WebBrowser.

Pero vayamos por partes, ya que el proceso no es muy intuitivo, debido a cómo se graban los datos mediante el método Save de ADO en formato XML.

Proceso de construcción de un convertidor de recordsets de ADO en ficheros XML

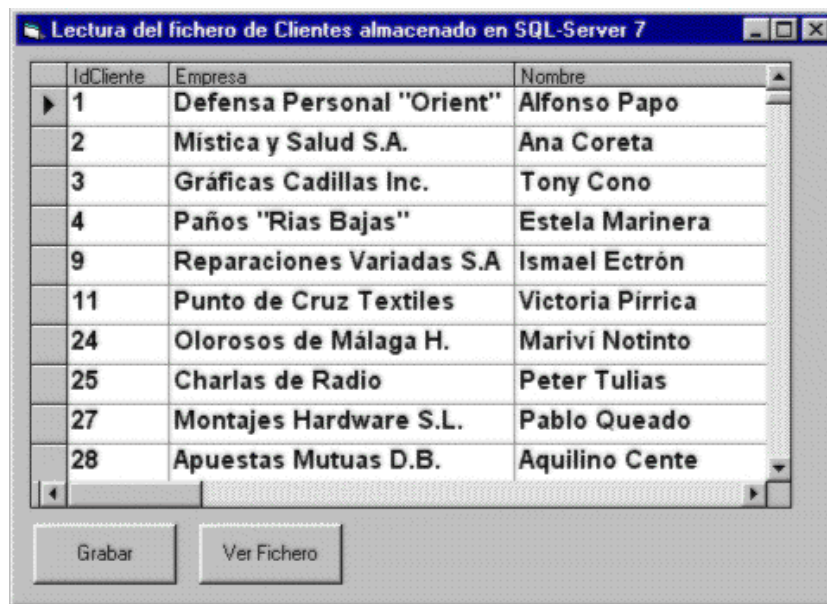
1º Paso: Verificar el formato producido por el recordset de ADO tras la grabación: Para ello, construimos un mecanismo estándar de acceso a una base de datos mediante ADO y mostramos el resultado en un control DataGrid para comprobar que los datos originales son correctos.

```
Const CadConex As String = "Provider=SQLOLEDB.1;Persist Security Info=False;User ID=sa;Initial Catalog=GesPedidos;Data Source=MARINO"
Dim Conex As New ADODB.Connection, Rs As New ADODB.Recordset
Private Sub Form_Load()
    Conex.ConnectionString = CadConex
    Conex.CursorLocation = adUseClient
    Conex.Open
    Rs.Open "Clientes", Conex, adOpenStatic, adLockOptimistic, adCmdTable
    Set DataGrid1.DataSource = Rs
End Sub
```

Código Fuente 49. Código fuente en Visual Basic para la conexión con el servidor SQL-Server

Mediante el Código Fuente 49, y supuesto que sean válidos los nombres del Servidor de datos y de la Base de Datos a acceder, obtendremos una primera salida del formulario con la información que deseamos mostrada en el DataGrid (téngase en cuenta que no hemos procedido a ninguna selección de los datos, se ha escogido la tabla completa, y para ello se le ha indicado como último parámetro al

Recordset el valor adCmdTable). Lo de menos ahora es el conjunto de registros obtenidos, lo que nos importa es que el objeto RS (el recordset) puede grabar en disco esa información con una formato XML.



IdCliente	Empresa	Nombre
1	Defensa Personal "Orient"	Alfonso Papo
2	Mística y Salud S.A.	Ana Coreta
3	Gráficas Cadillas Inc.	Tony Cono
4	Paños "Rias Bajas"	Estela Marinera
9	Reparaciones Variadas S.A	Ismael Ectrón
11	Punto de Cruz Textiles	Victoria Pirrica
24	Olorosos de Málaga H.	Marivi Notinto
25	Charlas de Radio	Peter Tulias
27	Montajes Hardware S.L.	Pablo Queado
28	Apuestas Mutuas D.B.	Aquilino Cente

Figura 19. Vista de datos de Clientes en un DataGrid

A continuación, en el botón cmdGrabar, asignamos el Código Fuente 50.

```
Private Sub cmdGrabar_Click()

    If Dir("C:\Clientes_t.xml") <> "" Then Kill "C:\Clientes_t.xml"
    Rs.Save "C:\Clientes_t.xml", adPersistXML

End Sub
```

Código Fuente 50. Código para grabar un recordset con formato XML

Observe el lector el segundo parámetro del método Save. Es lo que indica a ADO que debe utilizar el formato estándar XML en la grabación. Como consecuencia, se crea un fichero con un Schema, y los datos almacenados en formato XML dentro de elementos **z:row** asignados a atributos de esos elementos con el nombre del campo. El fichero de salida tiene el aspecto de la Figura 20.

Como vemos, ADO construye un XML-Schema a partir de los metadatos del fichero y si continuamos la visualización, comprobamos que los datos se han almacenado en atributos de cada elemento **z:row** (Figura 21). De esa forma, tal y como comentábamos antes, se evita tener que realizar comprobaciones de valor, ya que los atributos se almacenan por definición como CDATA, y no es preciso realizar el análisis de caracteres.

El inconveniente, a priori, parece estar en el hecho de que ADO utiliza una serie de definiciones de espacios de nombres (**s**, **rs** y **z**), para hacer referencia respectivamente (sin ambigüedades posibles) al Schema, Recordset y Fila (row) que constituyen los datos guardados. Nuestra labor será la de eliminar ese Schema, y construir un DTD a partir de la información del Recordset, que luego podamos mostrar con facilidad en el navegador.

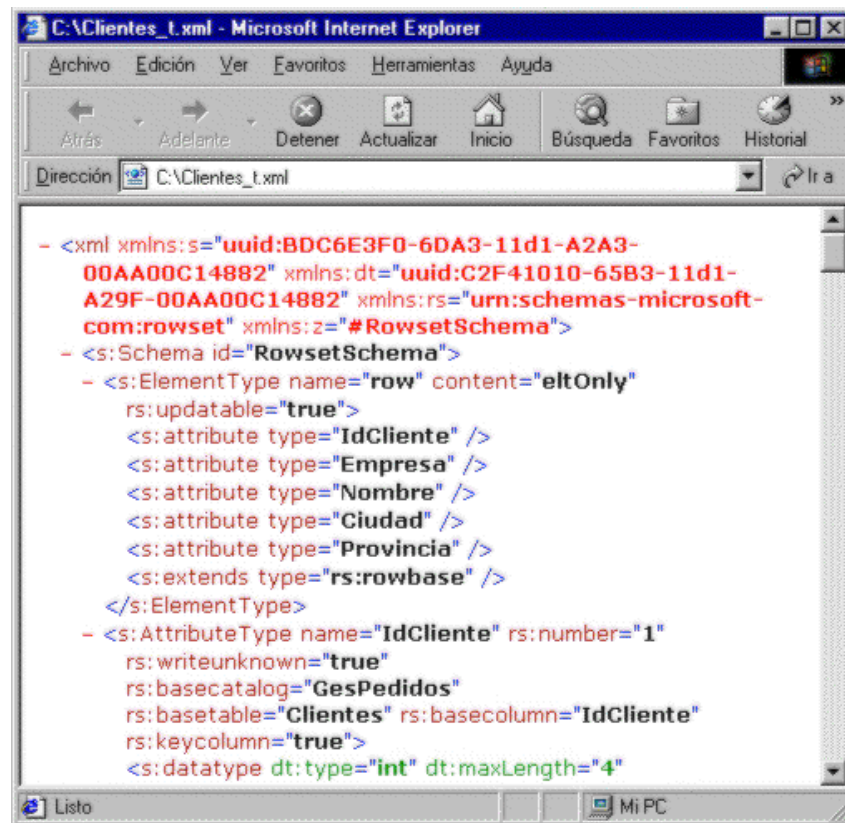


Figura 20. Fragmento del XML-Schema construido por ADO 2.1 a partir del recordset de Clientes

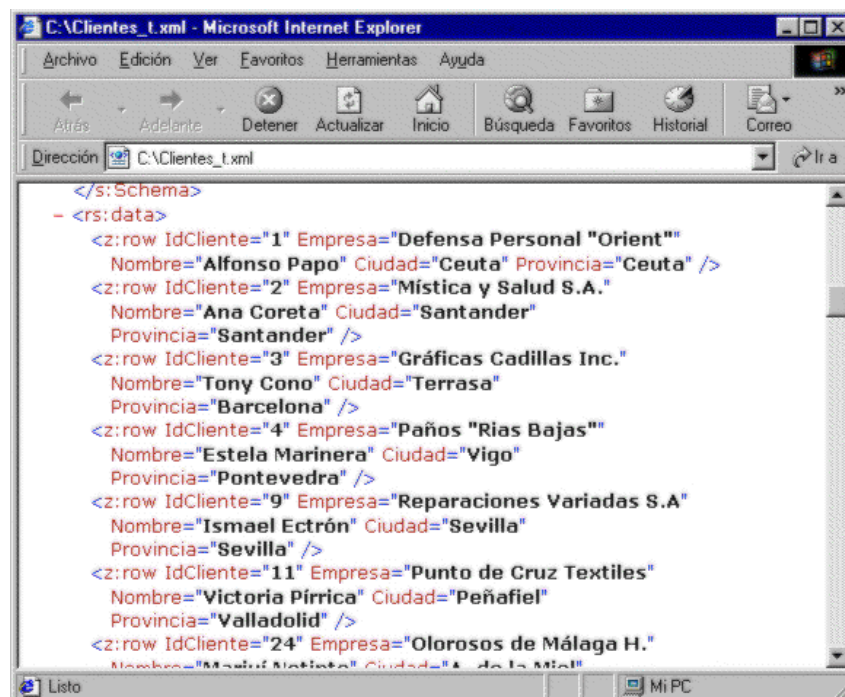


Figura 21. Almacenamiento de datos en atributos generado por el recordset de ADO

El Código Fuente 51, corresponde a la primera parte del código del botón **Ver Fichero**, mediante el cual construimos el nuevo DTD. El truco consiste en que vamos a leer el fichero de datos utilizando

formato de texto plano, mediante un control RichTextBox, y crearemos dinámicamente un nuevo DTD para asociarlo a la propiedad Text de ese control, y poder grabar la información modificada sin tener que recurrir a rutinas de proceso a bajo nivel que nos harían la explicación más compleja de lo que, por sí, ya es.

Por tanto el código que sigue, ilustra 3 pasos:

1. Carga del fichero generado anteriormente en un control RichTextBox
2. Eliminación del Schema generado
3. Creación de un DTD “ad hoc”, sabiendo que los datos van a estar almacenados en atributos y no en elementos, como hasta ahora hemos estado utilizando.

```
Private Sub cmdVerFic_Click()

    On Error GoTo Errores
    Dim NumCar As Integer, Campo As ADODB.Field, _
        DTDatributos As String, DTD As String
    Load Form2
    'Primero cargamos el fichero en el formato generado por ADO
    Form2.RT1.FileName = "C:\TDatos.xml"
    'Ahora vamos a sustituir la definición del Schema por un
    'DTD, asignando los datos a atributos del elemento row (fila),
    'que, a su vez es hijo del nodo raíz (RS, o Recordset)
    'Seleccionamos todo el texto de definición de Schema
    NumCar = Form2.RT1.Find("</s:Schema>")
    Form2.RT1.SelStart = 0

    Form2.RT1.SelLength = NumCar + Len("</s:Schema>")
    Form2.Show
    SendKeys vbNewLine & "<?xml:stylesheet type='text/xsl' href='Presentacion.xsl'
?>", 1
    '...y le sustituimos por declaraciones estándar y el
    'DTD. Asumimos que existe un fichero de presentación en
    'el mismo directorio del fichero de datos, con el nombre
    'Presentacion.xsl. Previamente hay que obtener la parte
    'no estándar del DTD: la que define los campos del recordset
    'como atributos del elemento row
    DTD = "<!DOCTYPE rs [" & vbNewLine & _
        "<!ELEMENT rs (row+)>" & vbNewLine & _
        "<!ELEMENT row EMPTY>" & vbNewLine

    For Each Campo In Rs.Fields
        DTDatributos = DTDatributos & "<!ATTLIST row " & _
            Campo.Name & " CDATA #IMPLIED>" & vbNewLine
    Next
    DTDatributos = DTDatributos & "]">
    DTD = "<?xml version='1.0' encoding='UTF-8'?>" & vbNewLine & _
        DTD & DTDatributos
    'Queda sustituir el espacio de nombres de Rs
    Form2.RT1.Text = Replace(Form2.RT1.Text, "rs:data", "rs")
    '...eliminar la última etiqueta </XML>
    Form2.RT1.Text = Replace(Form2.RT1.Text, "</xml>", "")
    'y eliminar las referencias al espacio de nombres
    'creado para las filas (z:). Grabamos el nuevo fichero
    'modificado
    Form2.RT1.Text = Replace(Form2.RT1.Text, "z:row", "row")
    Form2.RT1.Text = DTD & Form2.RT1.Text
    Form2.RT1.SaveFile "C:\TDatos.xml", 1
```

Código Fuente 51. Primera parte del código fuente del botón Ver Fichero

Después de todo este trabajo, obtendremos un fichero como nos muestra el control RichTextBo, Figura 21.

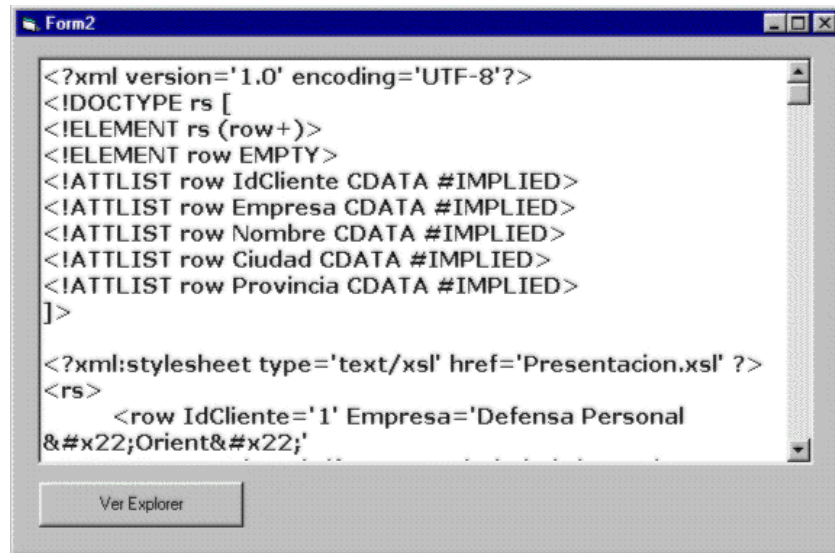


Figura 21. Fichero XML con el nuevo DTD generado por la rutina anterior

Todavía nos falta algo importante, la presentación. Como podemos utilizar la misma técnica para construir un fichero XSL de presentación, le vamos a añadir al Código Fuente 50, otro fragmento de código (Código Fuente 52) para que realice ese trabajo por nosotros.

```
'Ahora generamos un fichero de presentación estándar XSL
Form2.RT1.Text = "<?xml version='1.0' encoding='UTF-8'?>" & vbNewLine & _
  "<xsl:stylesheet xmlns:xsl='http://www.w3.org/TR/WD-xsl'>" & _
  vbNewLine & "<xsl:template match='/'>" & vbNewLine & _
  "<HTML><BODY>" & vbNewLine & _
  "  <xsl:for-each select='//row'>" & vbNewLine & _
  "    <SPAN STYLE='font-family:Arial; font-size:11Pt;color:BLUE'>"

Dim CadAtributos As String
For Each Campo In Rs.Fields
  CadAtributos = CadAtributos & _
    "<xsl:value-of select='@" & Campo.Name & "'/>" & vbNewLine
Next
CadAtributos = CadAtributos & _
  "<HR/></SPAN>" & vbNewLine & "</xsl:for-each>" & vbNewLine & _
  "</BODY></HTML>" & vbNewLine & "</xsl:template>" & vbNewLine & _
  "</xsl:stylesheet>"
Form2.RT1.Text = Form2.RT1.Text & CadAtributos
Form2.RT1.SaveFile "C:\Presentacion.xsl", 1
Exit Sub

Errores:
  MsgBox "Error al procesar los ficheros", vbInformation
End Sub
```

Código Fuente 52. Fragmento restante con la rutina de generación del fichero XSL de presentación

Con esto, ya tenemos un fichero de datos *normalizado* y un fichero de presentación preparado para hacer referencia a los atributos de cada elemento **<row>**, presente en los datos. El código fuente de nuestro fichero de presentación se muestra en la Figura 22 en el control RichTextBox.

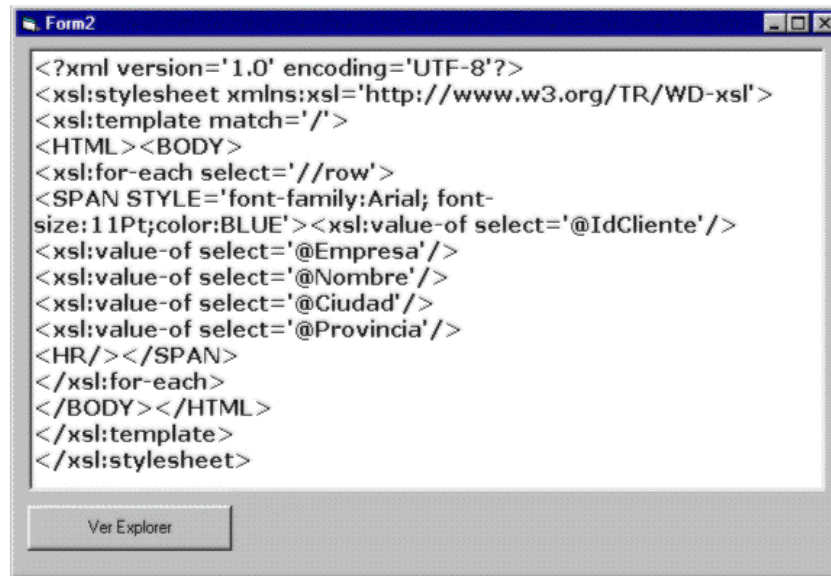


Figura 22. Fichero XSL generado por el Código Fuente 51

Finalmente, si utilizamos un tercer formulario con un control WebBrowser, podremos ver la salida tal y como se produciría en el navegador. Para ello, sólo tenemos que añadir el Código Fuente 53 al botón Ver Explorer, para que lance el tercer formulario y cargue el fichero XML generado, que llamará, a su vez, al fichero de presentación, para formatear la salida (Recuerde que hemos incluido una llamada al fichero “C:\Presentacion.xml”, dentro del fichero de datos XML).

```
Private Sub cmdVerExp_Click()
    Load Form3
    Form3.WebBrowser1.Navigate "file:///C:/Clientes_t.xml"
    Form3.Show
End Sub
```

Código Fuente 53. Código para llamar al formulario que contiene el control WebBrowser

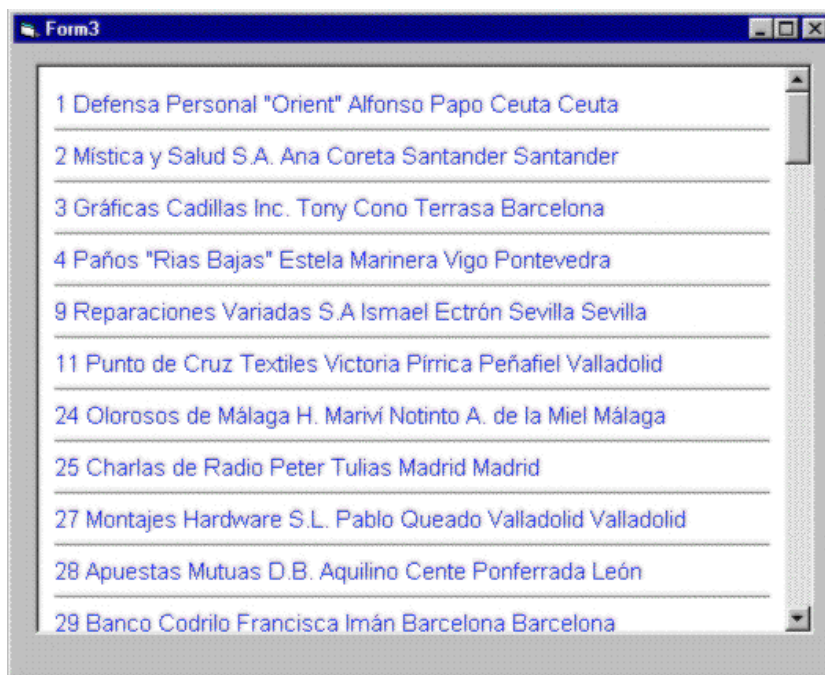


Figura 23. Salida final del fichero de datos XML usando la presentación generada

Conclusión

El trabajo de construir aplicaciones en las que los datos originales se encuentren en formato XML, requiere, como es lógico, mucho más trabajo que el expuesto en ésta breve reseña, especialmente para implementar la interfaz de forma similar a como hacemos con los datos provenientes de otros formatos. Lo interesante es que, gracias a ésta y otras librerías, existen mecanismos mediante los que, sin variar sustancialmente lo que hacemos, podemos implementar soluciones que utilicen éste nuevo estándar con todas las ventajas y utilizando mecanismos similares a los que veníamos usando hasta ahora.

Al lector interesado en éste tipo de soluciones, le remito a la referencia bibliográfica que encontrará en la última página y, en especial, a las obras de *Stephen Mohr* y *Paul Spencer*.

Para concluir, es interesante mencionar una característica de MSXML que no está disponible en todos los intérpretes y que –por el momento– tampoco figura en la recomendación del estándar DOM publicado por la W3C. Nos referimos al método **Save** del objeto **DOMDocument**, que permite hacer que un objeto que represente a un documento XML se convierta en un *objeto persistente*: esto es, que se pueda almacenar en disco una imagen del estado del árbol de elementos tal y como se encuentre en un momento dado, o incluso pasar como argumento al método **Save** un objeto **Response** de ASP, para que la información modificada por el cliente pueda actualizarse en el servidor.

Ejercicios

1. ¿Cómo se instancia una referencia a XML DOM?
2. ¿Depende el modo de instanciación del lenguaje de acceso?

3. ¿Qué sentencias se usan para instanciar XML DOM en *JavaScript*?
4. ¿Mediante que instrucción garantizamos que la carga de un documento XML se ha producido en su totalidad?
5. ¿Cómo podemos averiguar el número de nodos principales de un documento?
6. ¿Qué propiedades nos dan acceso al contenido de un nodo?
7. ¿Qué propiedades nos dan acceso al nombre de un nodo?
8. ¿Podemos acceder el nodo en formato XML?
9. ¿Qué propiedad muestra la existencia de nodos hijos?
10. ¿Qué dos propiedades utilizamos para movernos entre nodos del mismo nivel?

Recursos XML en Internet

En la actualidad están empezando a proliferar en Internet todo tipo de recursos relacionados con la tecnología XML. En la Tabla 12 podemos encontrar algunos de los más interesantes agrupados por categorías (El lector podrá encontrar información adicional sobre los estándares y también herramientas gratuitas y otros recursos en los siguientes sitios Web).

Sitios "Oficiales"	World Wide Web Consortium	www.w3.org
	Sitio XML de Microsoft	msdn.microsoft.com/xml
	Sitio XML de IBM	http://www.ibm.com/developer/xml/
Tutoriales	IBM (Castellano)	http://www.software.ibm.com/xml/education/tutorial-prog/overview.html
	Universidad de Málaga	http://face.el.uma.es/imasd/xml/xml.html
	WebMonkey	http://www.hotwired.com/webmonkey/
Revistas (Castellano)	RevistaWeb	http://www.revistaweb.com

Software gratuito	Universidad de Oslo (Noruega)	http://www.stud.ifi.uio.no/~larsga/linker/XMLtools.html
	Language Technology Group	http://www.ltg.ed.ac.uk/software/xml/
	IBM-alphaWorks	http://www.wdvl.com/Authoring/Languages/XML/IBM/
Editoriales	Wrox	http://webdev.wrox.co.uk/xml/
Webs sobre XML para desarrolladores:		http://www.xmlx.com http://www.schema.net

Tabla 12

Además pueden encontrarse un buen número de enlaces a otros sitios Web relacionados donde podrá obtener abundante información adicional, incluyendo artículos, programas editores *shareware* y *freeware*, etc.

LaLibreríaDigital.com

Si quiere ver más textos en este formato, visítenos en: <http://www.lalibreriadigital.com>.

Este libro tiene soporte de formación virtual a través de Internet, con un profesor a su disposición, tutorías, exámenes y un completo plan formativo con otros textos. Si desea inscribirse en alguno de nuestros cursos o más información visite nuestro campus virtual en: <http://www.almagesto.com>.

Si quiere información más precisa de las nuevas técnicas de programación puede suscribirse gratuitamente a nuestra revista *Algoritmo* en: <http://www.algoritmodigital.com>.

Si quiere hacer algún comentario, sugerencia, o tiene cualquier tipo de problema, envíelo a la dirección de correo electrónico lalibreriadigital@eidos.es.