Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Viana do Castelo

XML - Extensible Markup Language

Documento introdutório

Versão 1.0

Jorge Ribeiro **jribeiro@estg.ipvc.pt**

Índice

1. Introdução

Extensible Markup Language (XML) é linguagem de marcação de dados (meta-markup language) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. O XML também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via internet.

O XML permite a definição de um número infinito de tags. Enquanto no HTML, se as tags podem ser usadas para definir a formatação de caracteres e parágrafos, o XML provê um sistema para criar tags para dados estruturados.

Um elemento XML pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as tags XML são adotadas por intranets de organizações, e também via Internet, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados. Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado em um browser como o Internet Explorer de várias formas possíveis, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização.

2. Noções de HTML

Na internet atualmente quase todas as páginas se resumem em HTML (HyperText Markup Language). O termo *hypertext* é definido por textos que têm links para outros textos. Já o termo markup language define anotações para a estrutura de um texto. O design de documentos html tem duas características importantes:

1. Documentos html são feitos para prover estrutura lógica da informação destinada à apresentação de páginas da rede mundial de computadores.

2. A linguagem html contém um conjunto de tags com um número fixo para definir a estrutura do documento, e cada tag tem a sua semântica já definida. O CSS (Cascading Style Sheets) permite a separação da estrutura lógica da aparência da página. Mas, embora o layout possa ser separadamente definido no CSS, o html é destinado especificamente para hipertexto, e não para informação em geral!

2.1 Evolução do HTML

Essa linguagem foi desenvolvida em 1992 por Tim Berners Lee e Robert Caillau no CERN, que é o Centro Europeu de Pesquisas de Física de Partículas. O html é um exemplo do SGML (Standard Generalized Markup Language). Originalmente o html definia estritamente a

estrutura lógica de um documento, e não a sua aparência física. Mas, com a pressão dos usuários (principalmente da indústria), as versões posteriores do html foram forçadas a prover cada vez mais e mais controle da aparência do documento. Algumas datas importantes:

-1992: html foi definido

-1993: algumas definições físicas da aparência, tabelas, formulários e equações matemáticas (HTML+)

-1994: HTML 2.0 (padrão para as características principais) e 3.0 (uma extensão do HTML+, entendido como um rascunho de padrão).

-1995 e 1996: Netscape e Internet Explorer definem seus próprios padrões e surge o HTML 3.2 baseado nas implementações correntes.

-1997: O HTML 4.0 é desenvolvido separando a apresentação da estrutura com style sheets (folhas de estilo).

-1999: Definição do HTML 4.01 (suaves modificações da versão anterior).

-2000: O XHTML 1.0 é criado, o qual consiste de uma versão XML do HTML 4.01.

2.2 Comparações entre HTML e XML

HTML e XML são primos. Eles derivam da mesma inspiração, o SGML. Ambos identificam elementos em uma página e ambos utilizam sintaxes similares. Se você é familiar com HTML, também o será com o XML. A grande diferença entre HTML e XML é que o HTML descreve a aparência e a acções em uma página na rede enquanto o XML não descreve nem aparência e acções, mas sim o que cada trecho de dados é ou representa. Por outras palavras, o XML descreve o conteúdo do documento.

Como o HTML, o XML também faz uso de tags (palavras encapsuladas por sinais '<' e '>') e atributos (definidos com name="value"), mas enquanto o HTML especifica cada sentido para as tags e atributos (e frequentemente a maneira pela qual o texto entre eles será exibido em um navegador), o XML usa as tags somente para delimitar trechos de dados, e deixa a interpretação do dado a ser realizada completamente para a aplicação que o está lendo. Resumindo, enquanto em um documento HTML uma tag <p> indica um parágrafo, no XML essa tag pode indicar um preço, um parâmetro, uma pessoa, ou qualquer outra coisa que se possa imaginar (inclusive algo que não tenha nada a ver com um p como por exemplo autores de livros).

Os arquivos XML são arquivos texto, mas não são tão destinados à leitura por um ser humano como o HTML é. Os documentos XML são arquivos texto porque facilitam que os programadores "façam o debug" mais facilmente as aplicações, de forma que um simples editor de textos pode ser usado para corrigir um erro em um arquivo XML. Mas as regras de formatação para documentos XML são muito mais rígidas do que para documentos HTML. Uma tag esquecida ou um atributo sem aspas torna o documento inutilizável, enquanto que no HTML isso é tolerado. As especificações oficiais do XML determinam que as aplicações não podem tentar adivinhar o que está errado em um arquivo (no HTML isso acontece), mas sim devem parar de interpretá-lo e reportar o erro.

3. XML

3.1 Características da linguagem XML

3.1.1 Representação estruturada dos dados

O XML provê uma representação estruturada dos dados que mostrou ser amplamente implementável e fácil de ser desenvolvida.

Implementações industriais na linguagem SGML (Standard Generalized Markup Language) mostraram a qualidade intrínseca e a força industrial do formato estruturado em árvore dos documentos XML.

O XML é um subconjunto do SGML, o qual é optimizado para distribuição através da web, e é definido pelo Word Wide Web Consortium(W3C), assegurando que os dados estruturados serão uniformes e independentes de aplicações e fornecedores.

O XML provê um padrão que pode codificar o conteúdo, as semânticas e as esquematizações para uma grande variedade de aplicações desde simples até as mais complexas, dentre elas:

● Um simples documento.

●Um registro estruturado tal como uma ordem de compra de produtos.

●Um objeto com métodos e dados como objetos Java ou controles ActiveX.

●Um registro de dados. Um exemplo seria o resultado de uma consulta a bancos de dados.

●Apresentação gráfica, como interface de aplicações de usuário.

●Entidades e tipos de esquema padrões.

●Todos os links entre informações e pessoas na web.

Uma característica importante é que uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado sem a necessidade de reacionar o servidor. Dessa forma, os servidores tem menor sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

O XML é considerado de grande importância na Internet e em grandes intranets porque provê a capacidade de interoperação dos computadores por ter um padrão flexível e aberto e independente de dispositivo. As aplicações podem ser construídas e actualizadas mais rapidamente e também permitem múltiplas formas de visualização dos dados estruturados.

3.1.2 Separação entre dados e apresentação

A mais importante característica do XML se resume em separar a interface com o usuário (apresentação) dos dados estruturados. O HTML especifica como o documento deve ser apresentado na tela por um navegador. Já o XML define o conteúdo do documento. Por exemplo, em HTML são utilizadas tags para definir tamanho e cor de fonte, assim como formatação de parágrafo. No XML você utiliza as tags para descrever os dados, como exemplo tags de assunto, título, autor, conteúdo, referências, datas, etc...

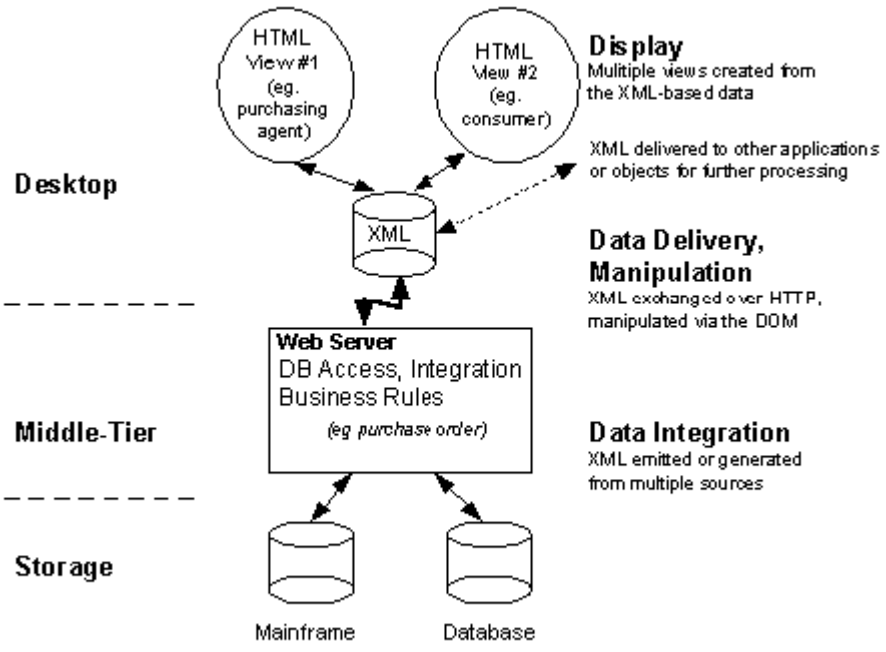

Figura 1 - Exemplo de aplicação Web em três níveis, a qual é flexível e permite a troca de dados entre mainframes e os clientes (desktops).

O XML ainda conta com recursos tais como folhas de estilo definidas com Extensible Style Language (XSL) e Cascading Style Sheets(CSS) para a apresentação de dados em um navegador. O XML separa os dados da apresentação e processo, o que permite visualizar e processar o dado como quiser, utilizando diferentes folhas de estilo e aplicações.

Essa separação dos dados da apresentação permite a integração dos dados de diversas fontes. Informações de consumidores, compras, ordens de compra, resultados de busca, pagamentos, catálogos, etc... podem ser convertidas para XML no middl- tier (espécie de servidor), permitindo que os dados fossem trocados online tão facilmente como as páginas HTML mostram dados hoje em dia. Dessa forma, os dados em XML podem ser distribuídos através da rede para os clientes que desejarem.

3.2 Definição conceptual do XML

3.2.1 Estrutura do documento

Um documento XML é uma árvore rotulada onde um nó externo consiste de:

- XML Prolog
- Elementos
- Atributos
- Character Data (CDATA)
- Processing Instructions
- Comments
- Entity References

XML Prolog

Declaração do XML. Primeira parte de qualquer documento XML bem formado.
Define a versão do XML utilizado assim como outra informação:

```
<?xml version="1.0"?>
<?xml version="1.0" encoding="iso-8859-1"?> <!–- Português-->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

standalone="yes" toda a validação necessária está no próprio documento

Pode conter também a Declaração do tipo de Documento <!DOCTYPE..>

Elementos

Correspondem a componentes básicos de um documento XML. São identificados por tags, como por exemplo:

```
<titulo>Os Lusíadas</titulo>
<autor>Eu</autor>
```

No entanto podem existir elementos vazios. Terminam com o símbolo "/", por exemplo: <p/>

O nome dos elementos devem começar por uma letra é case-sensitive: <p> é diferente de <P>

Pode conter texto ou outros elementos:

<titulo>Os Lusíadas <autor>Eu</autor></titulo>

Não se podem omitir ou sobrepor:

<titulo>Os Lusíadas<autor>Eu</titulo></autor> → Não é válido

Atributos

Correspondem a pares de nome-valor incluídos dentro dos elementos em que o valor deve estar sempre entre aspas "". Por exemplo:

<titulo nome="Os Lusíadas"/>

<frase autor="Eu">O Braga é o maior</frase>

Não se pode repetir o mesmo atributo num mesmo elemento:

<frase autor="Eu" autor="Tu"/> à Não é válido

Character Data (CDATA)

Este identificador permite inserir caracteres não "tratados" pelo parser. Caracteres do tipo: >, >, &, tags, etc..

A Syntax: <![CDATA[…]>

```
<MarkupSample>
        Isto é um exemplo errado de uso XMl com <tags>
</MarkupSample>
<MarkupSample>
        Isto é um exemplo correcto de uso XMl com <![CDATA[<tags>]]
</MarkupSample>
```

Entity References

Os valores que começam por "&" e terminam por ";"

```
> &gt;
< &lt;
& &amp;
‘ &apos;
“ &quot;
&#38; à referência a um caracter em decimal
&#x26; à referência a um caracter em hexadecimal
  à non-breaking space à   em HTML
```

Process Instructions (PI)

Indica para que alguma acção se execute. Indica-se por uma tag (em DOM) ou por um evento (em SAX). Syntax: <?....?>

```
<?xml version="1.0"?> à target=xml
<Autor>
        <?arquive 17> à target=arquive
        <Nome>…</Nome>
</Autor>
```

Comments

Comentários num documento XML(Semelhante ao HTML) .

Syntax: <!--....-->

Namespaces

Permitem trabalhar com vários vocabulários num mesmo documento. Têm o objectivo de:

- Evitar conflitos de nomes (ex. Variáveis e Módulos de um programa);
- Permitir que dois elementos com o mesmo nome possam existir embora em contextos diferentes;
- Trata-se de uma forma unívoca de identificar elementos e atributos num num documento XML: http://www.w3.org/TR/1999/REC-xml-names-19990114/

Exemplo:

```
<morada><rua>xxx</rua><number>27</number></morada>
<telef> <number>234567</number> </telef>
```

Neste exemplo poderia então ficar:

```
<morada><rua>xxx</rua><mor:number>27</mor:number></morada>
<telef> <tel:number>234567</tel:number> </telef>
```

Nestes casos mor e tel são Namespaces

Declaração de Namespace è < …xmlns:prefixo= >

Utilização de Namespace è < prefixo:nome >

O atributo xmlns:

- É necessário distinguir namespaces
- O valor de xmlns deve referenciar um URI de um Namespace
- Um documento pode conter Namespaces específicos ou por defeito;
- Os elementos/atributos sem prefixo estão associados ao Namespace por defeito

Exemplo:

```
<Tticket
xmlns="http://www.cdilearn.com/xml/troubleticket.dtd" <!--(por defeito)-->
xmlns:ticket="http://www.cdilearn.com/xml/ticket.dtd" ><!--(específico)-->
</Tticket>
```

3.2.2 Uma visão prática das tags

Um documento XML é um texto (em formato Unicode) com tags de marcação (markup tags) e outras informações. As markup tags denotam a seguinte estrutura:

```
...<bla attr="val" ...>...</bla>...
   |  |          | |
   |  |          | uma tag finalizadora de elemento
   |  |          o contexto do elemento
   |  um atributo com nome attr e valor val, com valores delimitados por ' ou "
```

uma tag inicializadora de elemento com nome bla

```
Notação para elementos vazios: ...<bla attr="val".../>...
```

Os documentos XML são sensíveis à letras maiúsculas e minúsculas.

Um documento XML é bem formatado quando segue algumas regras básicas. Tais regras são mais simples do que para documentos HTML e permitem que os dados sejam lidos e expostos sem nenhuma descrição externa ou conhecimento do sentido dos dados XML.

Documentos bem estruturados:

- têm casamentos das tags de início e fim.

- as tags de elemento tem que ser apropriadamente posicionadas

Os elementos não podem se sobrepor. Um exemplo de sobreposição é o seguinte:

```
<title>Descrição dos diveros modelos de carros<sub> da marca Ford
</title> Alexandre Manso</sub>
```

E, corrigindo o erro:

```
<title>Descrição dos diversos modelos de carros <sub> da marca Ford</sub>
 <author> Alexandre Manso</author> </title>
```

Caracteres especiais podem ser digitados usando referências de carcteres Unicode. Exemplo:

& = &.

Secções CDATA são formas alternativas de se usar dados de caracteres, como:

```
 <![CDATA[<greeting>Hello, world!</greeting>]]>
```

Informações adicionais:

```
<!-- comment -->
```

um comentário que será ignorado por todos os processadores.

```
<?target data...?>
```

uma instrução para um processador; target identifica o processador para o qual ela foi direcionada e data é a string contendo a instrução.

```
<!ENTITY name value>
```

declara uma entidade com um nome e um valor; expandida usando a referência entity: &name (entidades externas e referências de entidades de parâmetros são ignorados aqui).

```
<!ELEMENT ...>, <!ATTLIST ...>, ...
```

informações DTD (melhores alternativas são: DSD, XML Schema, que serão explicados posteriormente)

3.3 Principais benefícios da linguagem XML

O XML tem por objectivo trazer flexibilidade e poder às aplicações Web. Dentre os benefícios para os programadores e utilizadores temos:

- Pesquisas mais eficientes

- Desenvolvimento de aplicações Web mais flexíveis. Isso inclui integração de dados de fontes completamente diferentes, de múltiplas aplicações; computação e manipulação local dos dados; múltiplas formas de visualização e actualização granulares do conteúdo.

- Distribuição dos dados via rede de forma mais comprimida e escalável.

- Padrões abertos

Pesquisas mais eficientes

Os dados em XML podem ser unicamente "etiquetados", o que permite que, por exemplo, uma busca por livros seja feita em função do nome do autor. Actualmente, uma busca com o nome do autor poderia levar a qualquer site que tivesse referência a tal nome, não importando se fosse o autor do livro ou simplesmente um livro sobre o autor. Sem o XML é necessário para a aplicação de procura saber como é esquematizado e construído cada banco de dados que armazena os dados de interesse, o que é impossível. O XML permitiria definir livros por autor, título, assunto, etc..., o que facilitaria enormemente a busca.

Desenvolvimento de aplicações flexíveis para a Web

O desenvolvimento de aplicações Web em três camadas, ou three-tier, é altamente factível com o XML. Os dados XML podem ser distribuídos para as aplicações, objectos ou servidores

intermediários para processamento. Esses mesmos dados também podem ser distribuídos para o desktop (pc e similares) para ser visualizado em um navegador.

Integração de dados de fontes diferentes

Actualmente é praticamente impossível a procura em múltiplos bancos de dados e incompatíveis. O XML permite que tais dados possam ser facilmente combinados. Essa combinação seria feita via software em um servidor intermediário, estando os bancos de dados na extremidade da rede. Os dados poderiam ser distribuídos para outros servidores ou clientes para que fizessem o processamento, a agregação e a distribuição.

Computação e manipulação locais

Os dados XML recebidos por um cliente são analisados e podem ser editados e manipulados de acordo com o interesse do usuário. Ao contrário de somente visualizar os dados, os usuários podem manipulá-los de várias formas. Os recursos disponíveis do Document Object Model (DOM) permitem que os dados sejam manipulados via scripts ou outra linguagem de programação. A separação da interface visual dos dados propriamente ditos permite a criação de aplicações mais poderosas, simples e flexíveis.

Múltiplas formas de visualizar os dados

Os dados recebidos por um usuário podem ser visualizados de diferentes formas uma vez que o XML define somente os dados e não o visual. A interpretação visual poderia ser dada de várias maneiras diferentes, de acordo com as aplicações. Os recursos de CSS e XSL permitem essas formas particulares de visualização.

Atualizações granulares dos documentos

Os dados podem ser actualizados de forma granular, evitando que uma pequena modificação no conjunto de dados implique na busca do documento inteiro novamente. Dessa forma, somente os elementos modificados seriam enviados pelo servidor para o cliente. Atualmente, uma modificação em um item de dados acarreta na necessidade de actualização da página inteira. O XML também permite que novos dados sejam adicionados aos já existentes, sem a necessidade de reconstrução da página.

Fácil distribuição na Web

Assim como o HTML, o XML, por ser um formato baseado em texto aberto, pode ser distribuído via HTTP sem necessidade de modificações nas redes existentes.

Escalabilidade

Devido ao fato dos documentos XML separarem completamente os dados da forma com a qual são visualizados, autores de aplicações de visualização de dados podem torná-las muito poderosas e interativas, permitindo ao usuário visualizar os dados da forma que lhe agrade. Dessa forma, a interatividade, em termos, não dependeria tanto da comunicação cliente servidor, mas sim seria feita "offline", reduzindo o tráfego do link com o servidor.

Compressão

A compressão de documentos XML é fácil devido à natureza repetitiva das tags usadas para definir a estrutura dos dados. A necessidade de compressão é dependente da aplicação e da quantidade de dados a serem movidos entre clientes e servidores. Os padrões de compressão do HTTP 1.1 podem ser usados para o XML.

3.4 Exemplo de estruturas HTML e XML

Exemplo de receita em html e XML:

HTML	XML
Bolo de banana Miguel Furtado furtado@predialnet.com.br Sunday, 04 Jun 2000 O bolo de banana é feito com banana prata e possui um sabor maravilhoso. Pode ser servido quente ou frio. <table> <tr><td> 3 1/2 copos <td> de leite desnatado longa vida (leite em pó não é indicado). <tr><td> 2 colheres de sopa <td> de açúcar. <tr><td> 4 <td> bananas prata picadas em pedaços pequenos. <tr><td> 1 colher de chá <td> canela. <tr><td> 4 pedaços <td> noz moscada. </table> Combine tudo no liquidificador e bata até misturar bem. Leve ao forno por 20 minutos. Pronto. É só servir!Receitas relacionadas:	<recipe id="117" category="sobremesa"> <title> Bolo de banana </title> <author> <email> Miguel Furtado furtado@predialnet.com.br </email> </author> <date>Sunday,04 Jun 2000</date> <description> O bolo de banana é feito com banana prata e possui um sabor maravilhoso. Pode ser servido quente ou frio. </description> <ingredients> ... </ingredients> <preparation> Combine tudo no liquidificador e bata até misturar bem. Leve ao forno por 20 minutos. Pronto. É só servir! </preparation> <related url="#BoloChocolate">Bolo de Chocolate</related></recipe>

Bolo de Chocolate	

O exemplo mostra que:

1. As marcações são utilizadas puramente na estrutura lógica
2. Existe apenas uma única escolha do nível de detalhamento da marcação
3. Há a necessidade de uma espécie de "gramática" específica para a coleção de receitas em XML
4. Também é necessário style-sheet para definir a semântica da apresentação do documento.

3.5 Validação de documentos XML

Um documento XML pode ser um documento “Bem-formado” ou Válido. Um documento XML “Bem formado” segue as especificações do standard XML, ie, apresenta-se como “sintaticamente” correcto. Um documento XML Válido corresponde a um documento XML Bem formado que respeita as regras presentes num Document Type Definition (DTD) ou XML Schema, garantindo estrutura do documento e consistência de dados.

Um DTD – Document Type Definition, define:

- as marcas (tags) que podem ser utilizadas no documento XML;
– o número e a sequência de tags;
– os atributos de cada tag
– os tipo de valores que cada atributo pode vir a ter (opcional)

A sintaxe do DTD é definida por:

<!DOCTYPE RootElement (SYSTEM | PUBLIC)
ExternalDeclarations? [InternalDeclarations]?>
•RootElement: primeiro elemento da “árvore” XML
•SYSTEM: O ficheiro DTD está num ficheiro externo (DTD externo)
•PUBLIC: O ficheiro DTD pode residir algures num URL para ser partilhado
•ExternalDeclarations: URL associado ao DTD a usar
•InternalDeclarations: O DTD está definido dentro do doc. XML (DTD interno)

Um DTD é associado a um documento XML através de:
Externamente:

```
<?xml version="1.0"?>
<!DOCTYPE root_tag SYSTEM "file.dtd">
```

Dentro do próprio documento XML:

```
<!DOCTYPE name [ <!ELEMENT A (#PCDATA) …]>
```

A restante sintaxe é a seguinte:

Declaração de Elementos

<!ELEMENT nome_elemento (modelo_conteúdo)>

O conteúdo de um Elemento é definido da seguinte forma: <! ELEMENT revista (Conteúdo) >

(A, B, C) – Todos

(A | B | C) – Apenas um A, um B ou um C (não exclusivo)

(A* | B+ | C?) – Zero ou mais A; Vários B; C opcional

((A | B)*, C) -

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE revista SYSTEM "revista.dtd">
<revista>
    <A>qualquer coisa</A>
    <A>mais coisas</A>
    <C>tem de existir um C</C>
</revista>
```

<!ELEMENT revista ((A+, B?) | C*)>

Declarações de conteúdos vários

```
<!ELEMENT revista (#PCDATA | A | B | C )*>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B ANY>
<!ELEMENT C EMPTY>
<revista>Sem sub-elementos</revista>
<revista>
    Vários <A>valores A</A> para validar <B>valores B</B>
</revista>
```

Declarações de conteúdos ANY: Qualquer combinação de texto ou outros elementos, definidos no DTD.

Declarações de conteúdos TEXTO - #PCDATA

Pode ser utilizado sozinho ou no início de um modelo de conteúdo (combinado com outros elementos), sendo esse modelo uma sequência de elementos possíveis, ocorrendo zero ou mais vezes (*)

```
<!ELEMENT name(#PCDATA)>
<!ELEMENT paragraph (#PCDATA | Title | Footnote)*>
<!ELEMENT part_number (#PCDATA)*>
```

Declaração de Atributos de Elementos

<!ATTLIST nome_elemento definição_atributo>

Definição_Atributo= nome_atributo tipo_atributo classe val_defeito

```
<!ELEMENT revista EMPTY>
<!ATTLIST revista
        Cod ID #REQUIRED
        titulo CDATA #REQUIRED
        paginas (30 | 20 | 50) #REQUIRED
        existe CDATA “Sim”>
```

Declaração de Atributos: Tipos

CDATA	texto normal
Enumerate	conjunto de valores possíveis
ID	identificador único
IDREF	referência a um ID
IDREFS	mais do que uma referência
ENTITY	nome de uma entidade
ENTITIES	várias entidades
NOTATION	tipos para entidades
NMTOKEN	refere elemento XML
NMTOKENS	refere elementos XML

Exemplo 1:

```
<!ELEMENT revista EMPTY>
<!ATTLIST revista
        cod ID #REQUIRED
        titulo CDATA #REQUIRED
        paginas (30 | 20 | 50) #REQUIRED <!—enumerado obrigatório->
        existe CDATA “Sim” <!–- valor por defeito-->
ref IDREF >

<revista cod=“1” titulo=“o Maior” paginas=“30”/>
<revista cod=“2” titulo=“12 &gt; 10” paginas=“20”/> <!–- titulo = “12 > 10”-->
<revista cod=“3” titulo=“O Glorioso” paginas=“20” ref=“1”/>
```

Exemplo 2:

```
<!ELEMENT revista EMPTY>
<!ATTLIST revista
        cod ID #REQUIRED
        titulo CDATA #REQUIRED
        refs IDREFS
capa ENTITY>

<revista cod="Mac97" titulo="Os Lusíadas" refs=" OLA12 OLE34"/>
<revista cod="11" titulo="O Sporting está na corrida" capa="catedral"/>
```

Declaração de Notações

<!NOTATION gif PUBLIC "GIF">

Declaração de Entidades

<!ENTITY nome_entidade SYSTEM definição_entidade>

3.6 Aplicações do XML

Dentre os vários exemplos que podem ser dados estão:

● Aplicações em XHTML (**www.w3.org/TR/xhtml1**): Se resume na inserção de "data islands" XML em documentos html 4.0 formando o que se tem como padrão do W3C o XHTML.

"XMalização" de documento HTML 4.0 pelo W3C. Exemplo de um XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head><title>Hello world!</title></head>

<body><p>foobar</p></body> </html>
```

● Aplicações em CML (Chemical Markup Language, www.xml-cml.org):

Este é um exemplo de um trecho de um documento CML:

```
<molecule id="METHANOL">
<atomArray>
 <stringArray builtin="elementType">C O H H H H</stringArray>
<floatArray builtin="x3" units="pm">-0.748 0.558 -1.293 -1.263 -0.699 0.716</floatArray>
</atomArray>

</molecule>
```

● Aplicações em previsões de tempo (Weather Observation Markup Language)

● Aplicações nas descrições de conteúdo matemático com a linguagem MathML(Math Markup Language). Informações úteis estão disponíveis em: MathML 1.0.

● Aplicações de voz interativas (Extensible Phone Markup Language - XPML)

Uma lista mais completa de aplicações pode ser encontrada em www.oasis-open.org/cover/xml.html#applications

3.6 Ferramentas de utilização do XML

As ferramentas de utilização do XML podem dividir-se genericamente em três categorias:

● Editores de XML

- vi, emacs, ultraedit, notepad, xmlspy,...

● Browsers

– IE, Netscape, Mozilla, ...

● Validadores de XML

- http://www.webreference.com/xml/tools/
- http://www.hitsw.com/xml_utilites/

4. Bibliografia

● XML - A nova Linguagem da WEB, Pedro Coelho, FAC Editores.

●Professional XML, Bill Evjen, Kent Sharkey, Thiru Thangarathinam, Michael Kay, Alessandro Vernet, Sam Ferguson, Wrox.

● http://www.w3schools.com/xml/xml_whatis.asp

● http://www.zvon.org/xxl/XMLTutorial/General/book_en.html

Anexo 1 - XML Basics (part 1)

By icarus
July 23, 2001

Printed from DevShed.com
URL: http://www.devshed.com/Server_Side/XML/XMLBasic/XMLBasic1

Deconstructing The Silver Bullet

Unless you've spent the past few years down a rabbit hole, you've already heard about XML, the W3C's effort to create an extensible toolkit to store and manage different types of data. By defining a set of rules to organize collections of data, and then developing a set of technologies that can work with these organized collections, XML is a serious attempt to simplify the task of data management.

From the release of the first working draft of the XML specification in 1998, all the way through to its current incarnation, XML has been the subject of mass media hysteria, with technology pundits and business leaders alike proclaiming its virtues. XML, they say, is the silver bullet, the magic elixir that will cure all of humanity's woes...and help you make a profit in the bargain. Don't wait, they say; get on the XML bandwagon now and your customers will thank you for it.

Don't believe them.

XML is no silver bullet; it's a tool. A tool whose benefits lie primarily in the manner of its usage. Used correctly, XML and its related technologies can, indeed, make your life simpler, your processes more efficient, and perhaps even fatten your bottom line. Used incorrectly, it's simply a toy, a beautiful thing that you admire for a while and then go back to work. If you don't understand the basic concepts and principles of XML, and how to apply them to your requirements, your ability to exploit it will be, at best, limited.

Over the course of this series of articles, I will be focusing on core XML concepts - elements, attributes, namespaces, entities and the like - in the hope of offering a starting point to XML novices. In case you already know most of this stuff, fear not - at a later date, I will also be discussing the applications of XML and its related technologies - data exchange, transformations, linkages and more - together with more focused discussions of new and upcoming XML technologies.

This introductory article will discuss the origins and design goals of XML, the basic rules of XML markup, and how to use elements and attributes in an XML document. And class is now in session.

A Little History

XML, or Extensible Markup Language, is not new. In fact, it's a subset of SGML, the Standardized General Markup Language, modified for use on the Web. SGML was originally developed by Goldfarb, Mosher, and Lorie at IBM in 1969, as a way to structure legal documents; it has evolved over time into an international standard for representing textual data in system-independent format. Since SGML is overly complex for the requirements of the Web, XML has evolved as a modified (read: simpler) version of SGML, adapted specifically for use on the Web.

You might be thinking to yourself: isn't there already a universal language for the Web called HTML? And you'd be right to wonder...

While HTML is great for putting together Web pages, it doesn't offer any way to describe the data contained within those pages. As a formatting language, it doesn't offer any mechanism to define data structures within the document, thereby limiting its usefulness. The fact that it understands a limited set of tags - and even that frequently depends on which browser you're using - reduces its flexibility and makes it difficult to extend its usefulness to other applications.

XML was designed to avoid these disadvantages by creating a markup language which would be simple yet flexible, easy to use yet powerful enough to offer a variety of different applications. Briefly, the original design goals for XML (as stated in the W3C's XML 1.0 Recommendation) were: XML should be simple and easy to use.

XML should support a variety of different applications, by allowing users to develop their own markup.

XML documents should precisely follow certain formally-defined rules and principles.

XML documents should be human-legible and reasonably clear.

The Big Picture

They say that the whole is greater than the sum of its parts...and nowhere is this seen more clearly than with XML and its ancillary technologies. Over the past year and a half, the XML universe has grown by leaps and bounds to include many new technologies, most with hard-to-remember acronyms. Here's a quick list of the important ones, and how they fit into the larger picture:

XML Schema: XML Schema makes it possible to define the structure and format of "classes" of XML documents, providing more advanced features than those offered by the regular Document Type Definition (DTD). Find out more about it at http://www.w3.org/XML/Schema

XLink: XLink is a specification for linking XML data structures together, in much the same way as the hyperlinks available in HTML...although XLink allows for far more sophisticated types of links, including simultaneous links to more than one resource. Find out more about it at http://www.w3.org/XML/Linking

XPointer: XPointer is a specification for navigating the hierarchical tree structure of an XML document, and referencing elements, attributes and other data structures within the document. Find out more about it at http://www.w3.org/XML/Linking

XSL and XSLT: The Extensible Stylesheet Language (XSL) makes it possible to apply presentation rules to XML documents, and convert - or transform - them from one format to another. Find out more about it at http://www.w3.org/Style/XSL/

XHTML: The next version of HTML, XHTML combines the precision of XML markup with the easy-to-understand tags of HTML to create a more powerful and flexible language. Find out more about it at http://www.w3.org/MarkUp/

XForms: XForms offers a way to improve the current crop of HTML-based forms by separating the function of the form from its appearance, thereby making it possible to easily adapt a form for display on a variety of devices and systems. Find out more about it at http://www.w3.org/MarkUp/Forms/

XML Query: The XML Query effort is focused on creating a specification that makes it possible to query one or more XML document(s) and generate usable result data (in much the same way as SQL is used to retrieve database records.) Find out more about it at http://www.w3.org/XML/Query

XML Encryption: XML Encryption is a means of encrypting and decrypting XML documents, so as to secure it against unauthorized usage. Find out more about it at http://www.w3.org/Encryption/2001/

The list of XML-related technologies keeps increasing, and you should always refer to the W3C's Web site at http://www.w3.org/ for the latest information.

The Hammer And The Chisel

Before beginning any development effort with XML, you should make sure that you have the right development environment and tools.

The first - and most important - development tool is the XML editor. Since XML is a set of rules which allow for the description of textual data, XML documents can be created with any text editor (just like HTML.) On a UN*X system, both vi and emacs can handle XML documents, while Notepad remains one of my favourites under Windows. If you prefer something a little more user-friendly, take a look at XMLSpy, a powerful and full-featured XML editor, at http://www.xmlspy.com/, or XMetaL at http://www.xmetal.com/

Both Microsoft Internet Explorer 5.0 and Netscape Navigator 6.0 come with built-in XML support, and can read and display an XML document in a hierarchical tree view. Since most systems come with either or both of these installed, you don't need to look very far if you need a tool to simply display an XML document. In addition to these, both Opera and the W3C's Amaya browser now have support for XML documents.

It should be noted at this point that since one of the primary purposes of XML is to describe data - not present it - browser support is not an essential requirement for XML usage. Since XML is an open standard, it can be used to package data into structures that are easily transferable from one system to another. Consequently, you don't need to constrain yourself to a browser to validate XML data - James Clark's expat parser, at http://www.jclark.com/xml/expat.html, and Tim Bray's Lark parser, at http://www.textuality.com/Lark/, will both do the job for you.

In addition to the general-purpose tools listed above, there are a huge number of specialized little programs floating around the Web. As this series narrows its focus, I'll be identifying the tools most suited for specific applications; however, if you can't wait, drop by http://www.garshol.priv.no/download/xmltools/, a frequently-updated list of free XML software, and download to your heart's content.

Lights, Camera, Action!

OK, enough with the background - let's get our hands dirty. Consider the following XML document:

```
<?xml version="1.0"?>
```

```
<review>
		<genre>Action</genre>
		<title>X-Men</title>
		<cast>
			<person>Hugh Jackman</person>
			<person>Patrick Stewart</person>
			<person>Ian McKellen</person>
			<person>Famke Janssen</person>
		</cast>
		<director>Bryan Singer</director>
		<duration>104</duration>
		<year>2000</year>
		<body>Every once in a while, Hollywood takes a comic-book hero, shoots him
on celluloid, slaps in a few whiz-bang special effects and stands back to
see the reaction. Sometimes the results are memorable
(<title>Superman</title>, <title>Spiderman</title>, <title>Flash
Gordon</title>) and sometimes disastrous (<title>Spawn</title>, <title>The
Avengers</title>). Luckily, <title>X-Men</title> falls into the former
category - it's a clever, well-directed film that should please both
comic-book aficionados and their less well-read cousins.</body>
		<rating>4</rating>
</review>
```

As you can see, an XML document, like an HTML document, is simply an ASCII text file. This specific text file contains a recipe, broken up into different sections; each section is further "marked up" with descriptive tags to precisely identify the type of data contained within it.

An XML document may be either "well-formed" or "valid".

A well-formed document is one which meets the specifications laid down in the XML recommendation - that it, it follows the rules for element and attribute names, contains all essential declarations, and has properly-nested elements.

A valid document is one which, in addition to being well-formed, adheres to the rules laid out in a document type definition (DTD) or XML Schema. By imposing some structure on an XML document, a DTD makes it possible for documents to conform to some standard rules, and for applications to avoid nasty surprises in the form of incompatible or invalid data.

DTDs are essential when managing a large number of XML documents, as they immediately make it possible to apply a standard set of rules to different documents and thereby demand

conformance to a common standard. However, for smaller, simpler documents, a DTD can often be overkill, adding substantially to download and processing time.

Breaking It Down
Every XML document must begin with a declaration that states the version of XML being used; this declaration is also referred to as the "document prolog."

```
<?xml version="1.0"?>
```

This document prolog may also contain additional information, such as the document encoding, and whether the document is to be viewed in combination with external DTDs or other entities (as explained above, a DTD lays down the format for an XML document and can be used to verify whether or not it is valid.) Consequently, the document prolog can sometimes look like this,

```
<?xml version="1.0" standalone="yes" ?>
```

or this.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

The document prolog also contains a document type declaration, used to specify additional information about the document. This information typically includes the location of the DTD to use when validating the document (if there is one available), an optional list of entity declarations (more on this later), and the name of the root element of the document.

A document type declaration usually looks like this:

```
<!DOCTYPE rootElement DTDLocation

[

        entityDeclarations

]

>
```

A possible document type declaration for the movie review above might look like this:

```
<!DOCTYPE review SYSTEM "http://www.somedomain.com/review.dtd">
```

In this case, the document would be validated against the DTD located at http://www.somedomain.com/review.dtd

If entity declarations are present, this might be modified to read

```
<!DOCTYPE review SYSTEM "http://www.somedomain.com/review.dtd"

[

        <!ENTITY html "Hypertext Markup Language">

        <!ENTITY xml "Extensible Markup Language">

]
```

```
>
```

Entities will be discussed in detail in the second part of this article.

Simply Element-ary
The document prolog is followed by a series of "elements". An element, which is the basic unit of XML, consists of textual content (or "character data"), enhanced with descriptive tags (or "markup"). The boundaries of an element are defined by start and end tags, and may contain additional descriptive "attributes".

Here are some examples of XML elements:

```
<title>XML Basics</title>

<item>Nutcracker</item>

<dinosaur>Stegosaurus</dinosaur>
```

XML also allows for so-called empty elements - essentially, elements which have no content and therefore do not require a closing tag. Such elements are closed by adding a slash (/) to the end of their opening tag. For example,

```
<rule>Every sentence ends with a <period /></rule>
```

An element name must begin with a letter, optionally followed by more letters and numbers. For example,

```
<popeye>

<book>

<INCOME>
```

are all valid element names.

Element names are case sensitive - so

```
<me>
```

is different from

```
<Me>
```

or

```
<ME>
```

An element may contain only text,

```
<step>Garnish with lemon and chopped onions</step>
```

or a combination of text and other elements.

```
<sentence>The red <animal>wolf</animal> jumped over the blue
```

```
<vegetable>aubergine</vegetable></sentence>
```

In order to be well-formed, an XML document must contain at least one non-empty element. This outermost element is called the "root element" and, in turn, may contain other elements, nested in a hierarchical manner. In the first example above, the root element would be <review>...</review>.

Anyone For Chicken?
Let's look at another example:

```
<?xml version="1.0"?>

<recipe>

        <name>Chicken Tikka</name>

        <author>Anonymous</author>

        <date>1 June 1999</date>

        <ingredients>

                <item>

                        <desc>Boneless chicken breasts</desc>

                        <quantity>2</quantity>

                </item>

                <item>

                        <desc>Chopped onions</desc>

                        <quantity>2</quantity>

                </item>

                <item>

                        <desc>Ginger</desc>

                        <quantity>1 tsp</quantity>

                </item>

                <item>

                        <desc>Garlic</desc>

                        <quantity>1 tsp</quantity>

                </item>

                <item>

                        <desc>Red chili powder</desc>
```

```
                        <quantity>1 tsp</quantity>
                </item>
                <item>
                        <desc>Coriander seeds</desc>
                        <quantity>1 tsp</quantity>
                </item>
                <item>
                        <desc>Lime juice</desc>
                        <quantity>2 tbsp</quantity>
                </item>
                <item>
                        <desc>Butter</desc>
                        <quantity>1 tbsp</quantity>
                </item>
        </ingredients>
        <servings>
        3
        </servings>
        <process>
                <step>Cut chicken into cubes, wash and apply lime juice and salt</step>
                <step>Add ginger, garlic, chili, coriander and lime juice in a separate
                        bowl</step>
                <step>Mix well, and add chicken to marinate for 3-4 hours</step>
                <step>Place chicken pieces on skewers and barbeque</step>
                <step>Remove, apply butter, and barbeque again until meat is
               tender</step>
                <step>Garnish with lemon and chopped onions</step>
        </process>
</recipe>
```

Since each markup tag has a name which describes the data contained within it, it becomes possible to break up an unorganized document into structured, atomic parts. In the example above, the <author> tag identifies the data contained within it to be the name of the recipe author, while the <desc> and <quantity> tags are used to identify ingredients and their

respective quantities.

The textual content which appears between the opening and closing tags is referred to as "character data"...or, as the XML specification puts it, "all text that is not markup constitutes the character data of the document." Although this character data may contain alphanumeric characters or symbols, care should be taken to escape special characters like angle brackets and ampersands by replacing them with the corresponding hexadecimal representation or the strings

```
&lt;

&gt;

&amp;
```

respectively.

For example, while the following XML markup would generate an error,

```
to your left < is the yellow brick road
```

this would be absolutely fine.

```
to your left &lt; is the yellow brick road
```

Similarly, while

```
Barnes & Noble
```

would produce an error,

```
Barnes &amp; Noble
```

would have no trouble at all.

To Attribute Or Not To Attribute...
Elements can also contain attributes, which provide additional information about the element. Attributes are name-value pairs which appear within the start tag of an element and can be used to provide additional descriptive parameters or default values to the element. For example, the following XML snippet uses the attribute "sex" to provide additional data on the <person> element:

```
<cast>

<person sex="male">Hugh Jackman</person>

<person sex="male">Patrick Stewart</person>

<person sex="male">Ian McKellen</person>

<person sex="female">Famke Janssen</person>

</cast>
```

Attributes must always appear after the element name, and attribute names are case-sensitive. Attribute values must always be enclosed within quotation marks, and the same attribute should not be repeated twice within the same element. If your document is linked to a DTD, you can enforce rules on the types of values an attribute may and may not accept.

It should be noted that the line between attributes and elements is often very fine, since the two perform similar functions. For example, while it is perfectly valid for me to describe a <PERSON< using an attribute,

```
<person sex="male">Hugh Jackman</person>
```

I could achieve exactly the same effect by breaking the data down and assigning it to a series of elements.

```
<person>

<name>Hugh Jackman</name>

<sex>male</sex>

</person>
```

In other words - the decision as to whether to use an attribute or an element can sometimes be a tricky one, and needs to be made on a case-by-case basis. Most experts seem to agree that this is an implementation decision, and must be made keeping in mind the purpose for which the document is going to be used.

Valid reasons for using attributes over elements would include assigning an ID to a specific element,

```
<review id="6548450">...</review>
```

or describing characteristics of the element itself.

```
The <animal color="red">wolf</animal> jumped over the <vegetable

color="blue">aubergine</vegetable>
```

If you need to restrict attribute values to some pre-defined options, you can use a DTD to specify a list of allowed and default values, thereby cutting down on the possibility of errors and incompatible data.

If you're interested in a detailed discussion and debate of this issue, you should make it a point to visit http://xml.coverpages.org/elementsAndAttrs.html, which has some interesting comments and opinions by experts in the field on this very topics.

That's about it for the moment. In the next article, I'm going to continue this discussion of basic XML concepts with a look at entities, namespaces, and processing instructions - so make sure that you don't miss that one. Until then...stay healthy!

Anexo 2 - XML Basics (part 2)

By icarus
July 28, 2001

Printed from DevShed.com
URL: http://www.devshed.com/Server_Side/XML/XMLBasic/XMLBasic2

Tell Me More
In the first part of this article, I examined the need and rationale for XML, together with a brief look at the rapidly-increasing number of XML-related technologies. I discussed the basic structure and components of an XML document, played with the document prolog, and spent some time explaining how elements and attributes work. I also explained the difference between well-formed and valid XML, and demonstrated how the document prolog can be used to link an XML document to a DTD.

In this concluding article, I'll be examining some of the other things that go into an XML document, including CDATA, processing instructions, namespaces and entity references. Don't even think about going anywhere!

Splitting Up

First up, CDATA. As explained in the previous article, the XML specification considers all text enclosed within tags to be character data. There is one important exception to this - CDATA blocks.

CDATA blocks are document sections explicitly marked as not containing markup, and are hence treated as character data by the parser. These blocks can contain pretty much anything - strings, numbers, symbols, ancient Egyptian hieroglyphics - and will be ignored by the parser.

A CDATA block typically begins with

```
<![CDATA[
```

and ends with

```
]]>
```

with the data enclosed within the two. Here's an example:

```
<?xml version="1.0"?>

<manual>

        <function>split(str, pattern)</function>

        <description>Split a string <param>str</param> into component parts on the

basis of <param>pattern</param></description>
```

```
		<example>
		<![CDATA[
		<?
		split("apple, vanilla, orange", ",");
		?>
		]]>
		</example>
</manual>
```

CDATA blocks make it easy to add large blocks of text (including text containing special characters, symbols or program code) to an XML document, yet have the parser treat it as regular character data. And so, while a parser might choke on this,

```
<?xml version="1.0"?>
<secret_message>
		<from>Our man in Paris</from>
		<to>Director, Special Operations</to>
		<coded_body_text>
		12637 0%%348 83483 89238 82383 10341 0*049 27216 02039 84585 18127 45759
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461 40149 7^&291 21271
46101 42356 74(@1 4!128 47353 #511~ 473~7 12942 38#53 45628
		</coded_body_text>
</secret_message>
```

it will be absolutely fine with this.

```
<?xml version="1.0"?>
<secret_message>
		<from>Our man in Paris</from>
		<to>Director, Special Operations</to>
		<coded_body_text>
		<![CDATA[
		12637 0%%348 83483 89238 82383 10341 0*049 27216 02039 84585 18127 45759
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461 40149 7^&291 21271
46101 42356 74(@1 4!128 47353 #511~ 473~7 12942 38#53 45628
```

```
            ]]>

            </coded_body_text>

</secret_message>
```

Obviously, you cannot include the ending sequence

```
]]>
```

within a CDATA block, as this would merely serve to confuse the parser. If you need to include this sequence within a CDATA block, it needs to be written as

```
]]&gt;
```


Eating Humble PI
In addition to character data, XML also allows document authors to include specific instructions or commands to the processing application within the document. These instructions are referred to as "processing instructions", or PIs. PIs are not part of character data; instead, when an XML parser encounters a PI, it simply hands it over to the calling application, which has the option of using it (if it recognizes it) or ignoring it (if it doesn't.)

Every PI includes a target - this is the string used to identify the application to which the instruction is directed - followed by some data. This target-and-data combination is enclosed with <?...?> tags, as demonstrated by the following example:

```
<?xml version="1.0"?>

<directory>

            <category>Online Shopping<?rating popular?></category>

                        <url>http://www.amazon.com</url>

                        <desc>Amazon.com, the planet's foremost e-tailer<?link_with_ad
?></desc>

                        <url>http://www.cdnow.com</url>

                        <desc>CDNow.com, for all your music</desc>

                        <url>http://www.bn.com</url>

                        <desc>Barnes & Noble's online bookstore</desc>



</directory>
```

This data will be used by the XML application - for example, the first PI could indicate that the category be marked as "popular", while the second could link the item description with an

advertisement.

If you take a look at the document prolog (discussed in the last article), you'll see that the first line in any XML document,

```
<?xml version="1.0"?>
```

is actually a processing instruction.

XML And Alcohol

You've already seen how XML allows you to use descriptive tags to mark up text in a document. These tags are usually free-form; a document author has complete freedom to name these tags anything he or she desires. And while this flexibility is one of the reasons for XML's popularity, it's a double-edged sword, because it begs the question: what happens if tag names in different documents clash with each other?

An example might help to make this clearer. Let's suppose that I decided to encode my stock portfolio as an XML document. Here's what it might look like:

```
<?xml version="1.0"?>

<portfolio>

        <stock>Cisco Systems</stock>

        <stock>Nortel Networks</stock>

        <stock>eToys</stock>

        <stock>IBM</stock>



</portfolio>
```

And now let's suppose that Tom, my next-door neighbour and the proud owner of his own computer store, hears about XML, gets really excited, and assigns some of his employees to the task of encoding his store's inventory into XML. Here's what his XML document might look like:

```
<?xml version="1.0"?>

<inventory>

        <category>Mice</category>

        <item>Mouse C106</item>

        <vendor>Logitech</vendor>

        <stock>100</stock>

        <category>Handhelds</category>

        <item>Visor Deluxe</item>
```

```
        <vendor>HandSpring</vendor>

        <stock>23</stock>

        <category>MP3 players</category>

        <item>Nomad</item>

        <vendor>Creative</vendor>

        <stock>2</stock>

</inventory>
```

Finally, let's suppose that Tom and I get together for a drink, tell each other about our XML experiments and (in a moment of tequila-induced clarity) decide to put XML's capabilities to the test by combining our two documents into one. However, since both documents include a tag named

```
<stock>
```

whose meaning is entirely dependent on its context, it's pretty obvious that our attempt at integration will fail, since an XML application would have no way of telling whether the data enclosed between <stock>...</stock> tags belonged to my portfolio or Tom's inventory.

It's precisely to avoid this kind of ambiguity that the XML specification now provides for namespaces. Namespaces are a way to uniquely identify specific elements within an XML document. This is accomplished by assigning a unique prefix to an element, thereby immediately associating it with a particular data universe and eliminating ambiguity.

The Name Game

Setting up a namespace is simple - here's the syntax:

```
<elementName xmlns: prefix="namespaceURL">
```

In this case, the prefix is the unique string used to identify the namespace; this is linked to a specific namespace URL.

A namsepace is usually declared at the root element level, although authors are free to declare it at a lower level of the tree structure too.

Once the namespace has been declared within the document, it can be used by prefixing each element within that namespace with the unique namespace identifier. Take a look at my revised stock portfolio, which now uses a "mytrades" namespace to avoid name clashes.

```
<mytrades:portfolio

xmlns:mytrades="http://www.somedomain.com/namespaces/mytrades/">

        <mytrades:stock>Cisco Systems</mytrades:stock>

        <mytrades:stock>Nortel Networks</mytrades:stock>

        <mytrades:stock>eToys</mytrades:stock>
```

```
        <mytrades:stock>IBM</mytrades:stock>

</mytrades:portfolio>
```

And once Tom gets over his hangover, I guess he'd find it pretty easy to fix his document too.

```
<?xml version="1.0"?>

<toms_store:inventory xmlns:toms_store="http://www.toms_store.com/">

        <toms_store:category>Mice</toms_store:category>

        <toms_store:item>Mouse C106</toms_store:item>

        <toms_store:vendor>Logitech</toms_store:vendor>

        <toms_store:stock>100</toms_store:stock>


        <toms_store:category>Handhelds</toms_store:category>

        <toms_store:item>Visor Deluxe</toms_store:item>

        <toms_store:vendor>HandSpring</toms_store:vendor>

        <toms_store:stock>23</toms_store:stock>


        <toms_store:category>MP3 players</toms_store:category>

        <toms_store:item>Nomad</toms_store:item>

        <toms_store:vendor>Creative</toms_store:vendor>

        <toms_store:stock>2</toms_store:stock>


</toms_store:inventory>
```

In case you're wondering, the namespace URL is simply a pointer to a Web address, and is meaningless in practical terms; the XML specification doesn't really care where the URL points, or even if it's a valid link.

In case a single document contains two or more namespaces, adding namespace declarations and prefixes to every element can get kind of messy - as the following example demonstrates.

```
<?xml version="1.0"?>

<me:person xmlns:me="http://www.mywebsite.com/">

My name is <me:name>Huey</me:name>. I'm <me:age>seven</me:age> years old,

and I live in <me:address>Ducktown</me:address> with <rel:relationships
```

```
xmlns:rel="http://www.mywebsite.com/relationships/">my brothers

<rel:name>Dewey</rel:name> and

<rel:name>Louie</rel:name></rel:relationships>.

</me:person>
```

In such a situation, XML allows you to specify any one namespace as the default namespace, by omitting the prefix from the namespace declaration. Modifying the document above to make "me" the default namespace, we have

```
<?xml version="1.0"?>

<person xmlns="http://www.mywebsite.com/">

My name is <name>Huey</name>. I'm <age>seven</age> years old, and I live in

<address>Ducktown</address> with <rel:relationships

xmlns:rel="http://www.mywebsite.com/relationships/">my brothers

<rel:name>Dewey</rel:name> and

<rel:name>Louie</rel:name></rel:relationships>.

</person>
```

Namespaces need not be restricted to elements alone - attributes can use namespaces too, as the following example demonstrates.

```
<?xml version="1.0"?>

<mystuff:collection xmlns:mystuff="http://www.somedomain.com/mystuff"

xmlns:music="http://www.somedomain.com/music_genres">


        <mystuff:artist>Spears, Britney</mystuff:artist>

        <mystuff:title music:genre="Pop">Oops, I Did It Again!</mystuff:title>


</mystuff:collection>
```

An Entity In The Attic

XML entities are a bit like variables in other programming languages - they're XML constructs which are referenced by a name and store text, images and file references. Once an entity has been defined, XML authors may call it by its name at different places within an XML document, and the XML parser will replace the entity name with its actual value.

XML entities come in particularly handy if you have a piece of text which recurs at different places within a document - examples would be a name, an email address or a standard header or footer. By defining an entity to hold this recurring data, XML allows document authors to make global alternations to a document by changing a single value.

Consider the following simple example:

```
<?xml version="1.0"?>

<!DOCTYPE article

[

<!ENTITY copyright "This material copyright Melonfire, 2001. All rights

reserved.">

]>

<article>

        <title>XML Basics (part 2)</title>

        <abstract>A discussion of basic XML theory</abstract>

        <body>

        &copyright;

        Article body goes here

        &copyright;

        </body>



</article>
```

Entities come in two parts. First comes the entity definition, which always appears within the document type declaration at the head of the document (after the prolog). In this case, the entity "copyright" has ben defined and mapped to the string "This material copyright Melonfire, 2001. All rights reserved."

```
<!ENTITY copyright "This material copyright Melonfire, 2001. All rights

reserved.">
```

Once an entity has been defined, the next step is to use it. This is accomplished via entity references, placeholders for entity data within the document markup. Typically, an entity reference contains the entity name prefixed with either an ampersand (&) or a percentage (%) symbol and suffixed with a semi-colon(;), as below:

```
        <body>
```

```
&copyright;

Article body goes here

&copyright;

</body>
```

When a parser reads an XML document, it replaces the entity references with the actual values defined in the document type declaration. So this document

```
<?xml version="1.0"?>

<!DOCTYPE article

[

<!ENTITY copyright "This material copyright Melonfire, 2001. All rights

reserved.">

]>

<article>

<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>

&copyright;

Article body goes here

&copyright;

</body>


</article>
```

would look like this once a parser was through with it.

```
<?xml version="1.0" ?>

<title>XML Basics (part 2)</title>

<abstract>A discussion of basic XML theory</abstract>

<body>This material copyright Melonfire, 2001. All rights reserved. Article

body goes here This material copyright Melonfire, 2001. All rights

reserved.</body>
```

```
</article>
```

Note that entities must be declared before they are referenced, and must appear within the document type declaration. If a parser finds an entity reference without a corresponding entity declaration, it will barf and produce some nasty error messages.

XML comes with the following five pre-defined entities:

< - represents the less-than (<) symbol.

> represents the greater-than (>) symbol

' represents the single-quote (') symbol

&quote; represents the double-quote(") symbol

& represents the ampersand (&) symbol

Entities can contain XML markup in addition to ordinary text - the following is a perfectly valid entity declaration:

```
<!ENTITY copyright "This material copyright <link>Melonfire</link>,

<publication_year>2001</publication_year>. All rights reserved.">
```

Entities may be "nested"; one entity can reference another. Consider the following entity declaration, which illustrates this rather novel concept.

```
<!ENTITY company "Melonfire">

<!ENTITY year "2001">

<!ENTITY copyright "This material copyright &company;, &year;. All rights

reserved.">
```

Note, however, that an entity cannot reference itself, either directly or indirectly, as this would result in an infinite loop (most parsers will warn you about this.) And now that I've said it, I just know that you're going to try it out to see how much damage it causes.

Digging For Treasure

Entities come in a variety of flavours. They can broadly be divided into general entities and parameter entities. The examples you've seen above are general entities; since parameter entities are used only with DTDs, you don't need to worry about them for the moment.

Entities may be further classified into internal entities (entities defined within the document), external entities (entities defined in a separate file), and unparsed entities (entities which are not processed by the parser).

Most of the examples you've seen so far use internal entities - that is, the entity declaration and entity references are stored in the same physical document. XML also allows you to separate the entity declaration from the entity reference by storing it in a separate file, which comes in handy when the entity declaration contains a large block of text. Consider the following example:

```
<?xml version="1.0"?>
<!DOCTYPE article
[
<!ENTITY header "All source code copyright and proprietary Melonfire, 2001.
All content, brand names and trademarks copyright and proprietary
Melonfire, 2001. All rights reserved. Copyright infringement is a violation
of law. This source code is provided with NO WARRANTY WHATSOEVER. It is
meant for illustrative purposes only, and is NOT recommended for use in
production environments. Read more articles like this one at
<url>http://www.melonfire.com/community/columns/trog/</url> and
<url>http://www.melonfire.com/</url>">
]>
<article>
        <title>XML Basics (part 2)</title>
        <abstract>A discussion of basic XML theory</abstract>
        <body>
        &header;
        Article body goes here
        </body>

</article>
```

Since the entity contains a fairly large block of text, it may be more convenient to extract it and store it in a separate file, "header.xml". In that case, the example above would reduce to

```
<?xml version="1.0"?>
<!DOCTYPE article
[
<!ENTITY header SYSTEM "header.xml">
]>
<article>
        <title>XML Basics (part 2)</title>
```

```
        <abstract>A discussion of basic XML theory</abstract>

        <body>

        &header;

        Article body goes here

        </body>

</article>
```

In this case, the SYSTEM keyword is used to tell the parser the location of the file containing the replacement text for the entity.

Unparsed entities usually contain references to images, sound files or other binary data, and hence should not be processed by a parser (jeez, you think maybe that's why they're called "unparsed entities"?) Such entity declarations usually contain a link to the file (as with external entities) followed by an additional notation identifier which specifies the type of file.

In the following example, the NDATA keyword is used to tell the parser that the file being referenced is not to be processed in the usual manner; it is followed by a file type specification offering further information on the nature of the file.

```
<?xml version="1.0"?>

<!DOCTYPE article

[

<!ENTITY map SYSTEM "treasuremap.jpg" NDATA JPEG>

]>

<message>

To whoever finds this: here is a map showing the location of pirate

treasure on an island in the South Pacific.

&map;

Remember, X marks the spot.

</message>
```


The Man From IDIOT

Like HTML and most programming languages, XML also allows you to place comments within an XML document. A comment is simply an explanatory statement in plain English, intended to help others to understand and read your document. Comments are ignored by the parser, and are meant only for readability purposes - it's good programming practice to use them in your code.

Comments may appear anywhere within an XML document, and are similar to those used in HTML - a text string enclosed between <!-- and --> markers. Here's an example:

```
<?xml version="1.0"?>

<report>

        <headline>Alien Life On Earth, Says IDIOT Official</headline>

        <date> July 23, 2001</date>

        <place>Alaska</place>

        <reporter>Joe Cool</reporter>

<body>

<!-- who says you can't fool all of the people all of the time -->

In a not-unexpected turn of events, an IDIOT (I Doubt It's Out There)

official today confirmed reports of alien sightings in Area -10, the

coldest part of Northern Alaska, and again called on Pentagon officials to

either confirm or deny that the sightings were part of a decade-long

government project to breed alien lifeforms on Earth. IDIOT also claims to

have a map displaying the exact location of the alien "farm", and states

that it will be released to the press within the next forty-eight hours.

However, posing as an IDIOT, this intrepid reporter has successfully

obtained a copy of said map, reproduced below:

<!-- thanks, Mom -->

<map> </map>

</body>

</report>
```


Endgame

And that's about it for this crash course in XML theory. You now know enough to begin encoding XML documents on your own, as well as begin reading some of the more advanced material available on the subject. Here are a few links to get you started:

The W3C's XML specification, at http://www.w3.org/TR/2000/REC-xml-20001006

The Annotated XML specification, at http://www.xml.com/pub/a/axml/axmlintro.html
The W3C's Namespaces in XML specification, at http://www.w3.org/TR/1999/REC-xml-names-19990114/
Microsoft's XML Web site, at http://msdn.microsoft.com/xml/default.asp
XML.com, one of the best Web sites for articles and columns on XML, at http://www.xml.com/

XML articles and papers at http://xml.coverpages.org and http://www.xml-zone.com/articles.asp

Don't stray too far, though - what you've just learned is merely the tip of a very large iceberg, and over the next few weeks, I'll be delving into the next level of detail, discussing things like DTD design, XSL transformations and XLink data linkages.