XML Index

---

1. Introduction
2. XML Uses
3. XML Categories

Structure

1. Elements
2. Document Types
3. Document Structure
4. XML Content
5. Document Type Definition (DTD)
6. An Example
7. Element Attributes
8. Process Instructions
9. Extensible linking Language (XLL)

Use

1. Binding to HTML

Other

XSL

1. XSL Elements
2. XSL Example

DOM

3. DOM

Implementations

1. OSD

Appendices

1. Terms
2. Credits

The CTDP XML Tutorial Version 0.2.1, September 21 2001

XML Tutorial Introduction

Markup Language

If you have some familiarity with HTML, you have some concept of what markup language is. If you write a plain text file, it is composed of simple ASCII characters and nothing more. Whan a program (such as notepad) is used to display the file, all characters in the text file will be displayed using the same font size, type, and boldness. There are no special display characteristics to this type of file.

Markup languages, such as HTML or XML, allow special markup to be embedded with the rest of the text that will enable the program that displays the file to determine how to display the text. In this way, special text like headers may be centered, have a larger and bolder font, or specific display colors may be set. Also additional elements may be added to the file such as bulleted or numbered lists and tables.

Specifying Display Style

Markup languages use elements to set aside one area of content from other content. The display of these elements (such as color, size, and font type) may be determined within the markup file itself or outside the file using a style sheet. Normally, there is a predetermined set of display characteristics (default) for each element which may be modified locally or using style sheets. Authors are encouraged to separate the determination of display characteristics (style) from the markup file. This makes management of display style much easier but the separation is not required.

DTD

Markup languages normally require a Document Type Definition (DTD) which defines the elements that are allowed in the document. The DTD also defines how these elements may be used with relationship to each other. It will define how many elements and which elements may be included inside another element. The DTD is a text file written by a specific format to define the document. The DTD is based on the Standardized Generalized Markup Language (SGML). SGML is the parent of all markup languages. Although XML may use a DTD, it is not required for those documents that are considered "well formed". A well formed document follows a set of rules for XML and this subject is addressed in more detail later.

The DTD also defines other characteristics of the element such as whether or not it requires a beginning or ending tag along with various possible attributes of each element.

XML Definition

XML stands for extensible markup language. XML was developed around 1996 and is a subset of SGML. It's documents conform to SGML. XML was made less complicated than SGML to enable its use on the web. XML uses the ISO 10646 (Unicode) standard for encoding characters.

Previous Knowledge

Although not required, prior to reading this document, the reader is strongly encouraged to learn HTML and how to read and write DTDs by reading the HTML Guide and DTD Reference in the appropriate sections on this website. I believe understanding this document will be difficult without prior knowledge of DTDs and HTML, but those who desire to proceed without this knowledge may attempt it. Also the **information about Cascading Style Sheets (CSS) along with a list of CSS attributes and the types of elements they apply to are contained in the HTML Guide**.

XML Uses

When I first began to read about XML, I had to ask, "What are its practical uses?". I also wondered where it is actually being used currently and how can I actually use it. I was not aware of any tools that I could use to display XML. I did not know how I could see the results of my XML documents. However many modern web browsers today will display XML such as Internet Explorer version 5 or later.

Advantages

Since HTML has been so widely used and works so well (with some weaknesses) I had to wonder what the advantages of using XML are. A few are listed below:

- You can define your own elements and thereby support a much wider variety of information display. XML may be used to describe chemical structures, or other scientific or artistic data that cannot be readily displayed using HTML.
- Documents may be better organized into structures to allow for easier reference using and generating items like a table of contents.
- XML allows elements to be used to sort through database information readily.

Implementations

There are many implementations of XML which utilize their own DTD (Document Type Definition). Many have different purposes such as writing technical documentation, math formulas, sheet music, and so forth. The biggest drawback to these various implementations of XML is that for the most part, it seems that you will need to read the DTD or read documentation about the XML implementation in order to use it. If this is required, it would be like learning HTML for each XML implementation. Therefore there should be supporting programs that allow users to use XML implementations without having to learn each implementation. I think there are some programs that allow users to create XML files for some XML implementations, but I'm not sure which ones provide for this. My prediction is that XML implementations not supported by user friendly programs, will likely fail (and should) fail to become popular.

XML Categories

XML has several different categories that have slightly different purposes. They are listed below with descriptions.

- XML - The all encompassing category which includes the categories listed below. Specifically it refers to XML as used with a "well-formed" or valid document.
- XSL - Extensible Stylesheet Language is used to include additional style on XML pages. HTML code may even be embedded in the .xsl stylesheet file. This form of style sheet control is much more powerful and complicated than extensible style sheets.
- DSO - The data source object programming model is a technology provided by Microsoft which is used with small database support. It is a set of controls which can be used to allow a user to page through a table of information displayed on an HTML page which is linked to XML. This technology is not cross platform compatible. Defined byRFC2494.
- DOM - Document Object Model is a programming model that represents parts of an XML document as objects. Each object is thereby accessible by referring to the object and subobjects with periods between them. Each object supports specific attributes and methods which allows dynamic content using XML. The DOM is cross platform compatible. It allows scripts and programs to access and update the content and style of elements in a document. The DOM presents documents as a hierarchy of nodes.
- XLink
- RDF - Resource Description Framework

I have only mentioned some of the XML subcategories. More in-depth information may be found at The World Wide Web Consortium.

XML Elements

To create a document, it must contain elements. Lets assume that I want to create a document with the elements LAND, FOREST, TREE, MEADOW, GRASS. Here is how I would use these elements:

```
<LAND>
   <FOREST>
      <TREE>Oak</TREE>
      <TREE>Pine</TREE>
      <TREE>Maple</TREE>
   </FOREST>
   <MEADOW>
      <GRASS>Bluegrass</GRASS>
      <GRASS>Fescue</GRASS>
      <GRASS>Rye</GRASS>
   </MEADOW>
</LAND>
```

Each element is enclosed in <> brackets. The ending element has the '/' character before its name. As you can see, there is one element that contains all others, <LAND>. **XML requires one element that contains all others.** This single element, which in this case is "LAND", is called the root element. The FOREST element contains several TREE elements, and the MEADOW likewise contains several elements of GRASS. Each element that is contained in another ends in that same element and therefore each element is properly nested.

Elements that are included in another element are considered nested. The TREE elements in the above example are nested in the FOREST element. The FOREST element is the parent element to the TREE element and the TREE element is also called the sub-element to the FOREST element. These relationships hold true as you move up and down the element hierarchy. The FOREST and MEADOW elements are sub-elements to the LAND root element.

The below example is not well formed:

```
<LAND>
   <FOREST>
      <TREE>Oak</TREE>
      <TREE>Pine</TREE>
      <TREE>Maple
   </FOREST>
      </TREE>
```

```
    <MEADOW>
       <GRASS>Bluegrass</GRASS>
       <GRASS>Fescue</GRASS>
       <GRASS>Rye</GRASS>
    </MEADOW>
</LAND>
```

Element Tags

XML elements require both a beginning tag and an ending tag for all elements that have content. Elements with content may be written as:

<TREE>Very large Oak tree</TREE>

Elements with no content may be expressed as:

<NOTHING></NOTHING>

In shorthand it may be expressed as:

<NOTHING/>

Elements with no content may be used to display graphics and other material in the document.

Element Name Requirements

- Begins with a letter or underscore.
- The first character may be followed by any combination of letters, numbers, or other ASCII characters.
- Elements beginning with "XML" whether capatilized or not are reserved.

Element Content

Elements may contain:

- Nested elements - Other sub-elements.
- Processing instructions
- Characters - Normal text.
- CDATA sections - Used to enter text that contains special characters not displayed normally by the browser such as less than or greater than sign. These signs are used to enclose tags and are special characters. An example CDATA section:

```
<![CDATA[
The < and > characters are displayed normally here.
]]>
```

- Entity references - An entitity reference is precluded by a & sign. It is used to refer to a previously defined entity, much like a variable may be used in a program.
- Character references - References to characters that are not displayed normally in XMS such as the < or > characters. These characters are represented as &#60 and &#62 respectively and will be presented on the browser as the less than or greater than character they represent.
- Comments - Comments are included as shown below and may be placed anywhere except inside an element tag (markup).

```
<!-- This is a comment and is not displayed by the browser or renderer -->
```

XML Document Formation

Since XML does not require a Document Type Definition (DTD) there are two basic ways to create the document.

Types of XML Documents

1. Well Formed - The logical structure is not validated against the DTD. A well formed document follows a set of rules to qualify as "well formed".
2. Valid - Must have a DTD and obey all the rules. These rules are more stringent than rules for a well formed document. The document can be defined by a DTD which is either embedded inside the XML page or referenced as external to the page. Also you can write your own DTD or use a pre-existing DTD written by someone else.

Well Formed

The definition of "well formed" is:

- There must be one and only one top level element.
- All elements must have a starting and an ending tag with matching starting and ending names. Element names are case sensitive.
- Elements must be nested properly.

Valid

Valid documents must:

- Be well formed.
- Include a DTD.
- Follow the rules set by the DTD.

XML Structure

Document Parts

- Prolog
- Document Element (root element)

The Prologue

The prologue, equivalent to the header in HTML, may include the following:

- An XML declaration (optional) such as:

  <?xml version="1.0"?>

- A DTD or reference to one (optional). An example reference to an external DTD file:

  <!DOCTYPE LANGLIST SYSTEM "langlist.dtd">

- Processing instructions - An example processing instruction that causes style to be determined by a style sheet:

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

An XML Document

Therefore a complete well formed XML document may look like:

```
<?xml version="1.0"?>
<LAND>
   <FOREST>
      <TREE>Oak</TREE>
      <TREE>Pine</TREE>
      <TREE>Maple</TREE>
   </FOREST>
   <MEADOW>
      <GRASS>Bluegrass</GRASS>
      <GRASS>Fescue</GRASS>
      <GRASS>Rye</GRASS>
```

```
   </MEADOW>
</LAND>
```

The LAND element, above, is the root element.

The below document is not an XML document since it does not qualify by the rules of a well formed document. There is more than one top level element which disqualifies the document from being well formed.

```
<?xml version="1.0"?>
<FOREST>
   <TREE>Oak</TREE>
   <TREE>Pine</TREE>
   <TREE>Maple</TREE>
</FOREST>
<MEADOW>
   <GRASS>Bluegrass</GRASS>
   <GRASS>Fescue</GRASS>
   <GRASS>Rye</GRASS>
</MEADOW>
```

Defining Display

If the HTML document is not linked to a style sheet, the XML document will be displayed with tags included. The elements and tags may be color coded to aid in viewing the document. The document is displayed without tags according to the style sheet if a link to one is specified. The following document shows a document with a link to a cascading style sheet:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="xmlstyle.css"?>
<DATABASE>
   <TITLE>List of Items Important to Markup Languages</TITLE>
   <TITLE2>Languages</TITLE2>
      <LANGUAGES>SGML<LANGUAGES>
      <LANGUAGES>XML</LANGUAGES>
      <LANGUAGES>HTML<LANGUAGES>
   <TITLE2>Other</TITLE2>
      <OTHER>DTD<OTHER>
      <OTHER>DSSL<OTHER>
      <OTHER>Style Sheets</OTHER>
</DATABASE>
```

The below line, which is a part of the XML document above, is a processing instruction and is a part of the prolog.

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

The style sheet, "xmlstyle.css", may look like:

```
DATABASE
     { display: block }
TITLE
     { display: block;
       font-family: arial;
       color: #008000;
       font-weight: 600;
       font-size: 22;
       text-align: center }
TITLE2
     { display: block;
       font-family: arial;
       color: #000080;
       font-weight: 400;
       font-size: 20 }
LANGUAGES
     { display: block;
       list-style-type: decimal;
       font-family: arial;
       color: #000000;
       font-weight: 400;
       font-size: 18 }
OTHER
     { display: block;
       list-style-type: square;
       font-family: arial;
       color: #0000ff;
       font-weight: 200;
       font-size: 14 }
```

XML Content

The DTD, whether included as part of the XML file or external to the XML file is used to define content. It is used to determine the elements allowed in the file and which elements can be contained in other elements. It also describes the number of times specific elements may be contained in other elements. The document called "Document Type Definition (DTD)" on this website gives a more thorough explanation about how to read and construct a DTD. This document was written for SGML DTDs and the main difference lies in the fact that XML requires a beginning and ending tag for all elements. SGML does not have this requirement .

Element Declaration

Therefore in an SGML DTD an element tag for the <HR> element is:

```
<!ELEMENT HR - O EMPTY -- horizontal rule -->
```

If written for XML, the DTD would be:

```
<!ELEMENT HR EMPTY -- horizontal rule -->
```

The difference between the two examples is that the XML <HR> declaration does not define whether the element requires a closing tag as done with the "- O" text. This is because all elements in XML are required to have closing tags and therefore defining whether a closing tag is required is pointless.

XML DTD

In order to use DTDs with XML, the reader is encouraged to learn about the structure of DTDs. The document called "Document Type Definition (DTD)" on this website gives a more thorough explanation about how to read and construct a DTD than this document. The DTDs in this document are very basic and should, however, be easy to understand even for those who don't have familiarity with DTD's.

The DTD, whether included as part of the XML file or external to the XML file is used to define content. It is used to determine the elements allowed in the file and which elements can be contained in other elements. It also describes the number of times specific elements may be contained in other elements.

The DTD document on this website was written for SGML DTDs and the main difference lies in the fact that XML requires a beginning and ending tag for all elements. SGML does not have this requirement .

Element Declaration

Therefore in an SGML DTD an element tag for the <HR> element is:

```
<!ELEMENT HR - O EMPTY -- horizontal rule -->
```

If written for XML, the DTD would be:

```
<!ELEMENT HR EMPTY -- horizontal rule -->
```

The difference between the two examples is that the XML <HR> declaration does not define whether the element requires a closing tag as done with the "- O" text. This is because all elements in XML are required to have closing tags and therefore defining whether a closing tag is required is pointless.

XML Example

This is an example of an XML document that used an external DTD file and cascading style sheet (CSS) file.

The XML File

The following file is called "parts.xml".

```
<?xml version="1.0"?>
<!DOCTYPE PARTS SYSTEM "parts.dtd">
<?xml-stylesheet type="text/css" href="xmlpartsstyle.css"?>
<PARTS>
   <TITLE>Computer Parts</TITLE>
   <PART>
      <ITEM>Motherboard</ITEM>
      <MANUFACTURER>ASUS</MANUFACTURER>
      <MODEL>P3B-F</MODEL>
      <COST> 123.00</COST>
   </PART>
   <PART>
      <ITEM>Video Card</ITEM>
      <MANUFACTURER>ATI</MANUFACTURER>
      <MODEL>All-in-Wonder Pro</MODEL>
      <COST> 160.00</COST>
   </PART>
   <PART>
      <ITEM>Sound Card</ITEM>
      <MANUFACTURER>Creative Labs</MANUFACTURER>
      <MODEL>Sound Blaster Live</MODEL>
      <COST> 80.00</COST>
   </PART>
   <PART>
      <ITEM• inch Monitor</ITEM>
      <MANUFACTURER>LG Electronics</MANUFACTURER>
      <MODEL> 995E</MODEL>
      <COST> 290.00</COST>
   </PART>
</PARTS>
```

This file specifies the use of two external files.

- A DTD file called "parts.dtd". This is done on the second line:

  ```
  <!DOCTYPE PARTS SYSTEM "parts.dtd">
  ```

  The name after !DOCTYPE which is "PARTS" must be the same name as the encapsulating element in the XML file which is <PARTS> in this case.

- A CSS file called "xmlpartsstyle.css". This is done on the third line using a processing instruction:

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

The DTD File

The following file is called "parts.dtd".

```
<!ELEMENT PARTS (TITLE?, PART*)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST)+>
<!ATTLIST PART
   type (computer|auto|airplane) #IMPLIED>
<!ELEMENT ITEM (#PCDATA)>
<!ELEMENT MANUFACTURER (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT COST (#PCDATA)>
```

The root element is PARTS. The root element may contain no TITLE element or one TITLE element along with any number of PART elements. The PART element must contain one each of items ITEM, MANUFACTURER, MODEL, and COST in order. The PART element may contain an attribute called "type" which may have a value of "computer", "auto", or "airplane". The elements TITLE, ITEM, MANUFACTURER, MODEL, and COST all contain PCDATA which is parsed character data.

The Style File

The following file is used to set the style of the elements in the XML file. It is called "xmlpartstyle.css".

```
PARTS
     { display: block }
TITLE
```

```
     { display: block;
       font-family: arial;
       color: #008000;
       font-weight: 600;
       font-size: 22;
       margin-top: 12pt;
       text-align: center }
PART
     { display: block }
ITEM
     { display: block;
       font-family: arial;
       color: #000080;
       font-weight: 400;
       margin-left: 15pt;
       margin-top: 12pt;
       font-size: 18 }
MANUFACTURER
     { display: block;
       font-family: arial;
       color: #600060;
       font-weight: 400;
       margin-left: 45pt;
       margin-top: 5pt;
       font-size: 18 }
MODEL
     { display: block;
       font-family: arial;
       color: #006000;
       font-weight: 400;
       margin-left: 45pt;
       margin-top: 5pt;
       font-size: 18 }
COST
     { display: block;
       font-family: arial;
       color: #800000;
       font-weight: 400;
       margin-left: 45pt;
       margin-top: 5pt;
       font-size: 18 }
```

How it Displays on your browser

If you click on the following link and the loaded document does not display in color, your browser probably does not support XML. In this case, you should update your browser to the most recent version.

Click Here to see the PARTS.XML file (USE your browser's BACK button to return).

XML Element Attributes

An XML attribute may be included with element declarations. Attributes are name/value pairs that may be associated with the element. They may be used to control some characteristics of the element. In the markup document attributes are used with the following form:

```
<ELEMENTNAME Importance="minimal">
```

Rules for XML attribute names are the same as rules for XML element names. The attribute name may only appear once in the element start tag. The double or single quoted string may be used to delimit the attribute value. The attribute value may not contain the < character. The attribute value can only include the & character when using an entity or character reference.

The DTD determines which element attributes are required and what the defaults are. Read the DTD Reference manual on this website for more information about writing DTDs and specifying element attributes.

XML Process Instructions

XML process instructions are normally included in the XML document prolog, commonly thought of as the header in HTML. Processing instructions may be placed anywhere in the document so long as they are outside the element tags.

standalone="yes"

The rules for the names of process instructions are similar to the rules for element names, however in the case of process instructions there are some reserved xml process instructions such as "xml-stylesheet".

The process instruction syntax:

```
<? target instruction ?>
```

The *target* is the application name the instruction is meant for. It can be a reserved value such as "xml-stylesheet" or the name of an external application such as a script program name.

Extensible linking Language (XLL)

XLL formally was called XLink. XLL stands for Extensible Linking language and consists of the following parts:

- XPointer - It is built on XPath and it establishes a common system to specify node locations. It allows for locating data that is not a complete node.
- XLink - Advanced linking. It links to multiple destinations, is bi-directional and allows for links to be displayed on other documents
- XInclude - Merges XML infosets into one infoset.

Binding XML to HTML

Binding XML to HTML is used to display XML on an HTML page. The XML binding technique described on this page relies on proprietary Microsoft-addons which may only work for Microsoft Internet Explorer 5.0 and above. This binding is based on the Microsoft Data Source Object (DSO) programming model which allows scripts or data binding to be used to display an XML document from HTML.

To bind an XML file to an HTML file:

1. Use a line similar to the following:

   ```
   <XML ID="partsList" SRC="parts.xml"></XML>
   ```

2. Use the SPAN element or TABLE element to bind the XML file to the elements at the proper location

Here's the HTML code:

```
<XML ID="partsList" SRC="parts.xml"></XML>
<table DATASRC="#partsList">
<thead>
<th>Item</th><th>Manufacturer</th><th>Model</th><th>Cost</th>
<thead>
<tr>
<td><SPAN DATAFLD="ITEM"></SPAN></td>
<td><SPAN DATAFLD="MANUFACTURER"></SPAN></td>
<td><SPAN DATAFLD="MODEL"></SPAN></td>
<td><SPAN DATAFLD="COST"></SPAN></td>
</tr>
</table>
```

Here's the XML file, parts.xml.

```
<?xml version="1.0"?>
<!DOCTYPE PARTS SYSTEM "parts.dtd">
<?xml-stylesheet type="text/css" href="xmlpartsstyle.css"?>
<PARTS>
   <PART>
      <ITEM>Motherboard</ITEM>
```

```
      <MANUFACTURER>ASUS</MANUFACTURER>
      <MODEL>P3B-F</MODEL>
      <COST> 123.00</COST>
    </PART>
    <PART>
      <ITEM>Video Card</ITEM>
      <MANUFACTURER>ATI</MANUFACTURER>
      <MODEL>All-in-Wonder Pro</MODEL>
      <COST> 160.00</COST>
        </PART>
    <PART>
      <ITEM>Sound Card</ITEM>
      <MANUFACTURER>Creative Labs</MANUFACTURER>
      <MODEL>Sound Blaster Live</MODEL>
      <COST> 80.00</COST>
        </PART>
    <PART>
      <ITEM> 19 inch Monitor</ITEM>
      <MANUFACTURER>LG Electronics</MANUFACTURER>
      <MODEL> 995E</MODEL>
      <COST> 290.00</COST>
        </PART>
</PARTS>
```

Here's the DTD:

```
<!ELEMENT PARTS (PART*)>
<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST)>
<!ATTLIST PART
   type (computer|auto|airplane) #IMPLIED>
<!ELEMENT ITEM (#PCDATA)>
<!ELEMENT MANUFACTURER (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT COST (#PCDATA)>
```

Notice in the implementation, below, that the attributes specified by the XML file are overridden by the default attributes in HTML. None of the items are displayed using the color or font sizes specified by the CSS file that is referenced in the XML file, so that reference in the XML file is not effective.

Here's how it looks. If you can't see the contents of the table, your browser probably does not support XML or the Microsoft Data Source Object (DSO) programming model.

Item Manufacturer Model Cost

XSL Elements

A means of grouping elements

[msxml:script

xsl:apply-imports

xsl:apply-templates

xsl:attribute

xsl:attribute-set

xsl:call-template

xsl:choose

xsl:comment

xsl:copy

xsl:copy-of

xsl:decimal-format

xsl:document

xsl:element

xsl:fallback

xsl:for-each

xsl:if

xsl:import

xsl:include

xsl:key

xsl:message

xsl:namespace-alias

xsl:number

xsl:otherwise

xsl:output

xsl:param

xsl:preserve-space

xsl:processing-instruction

xsl:sort

xsl:strip-space

xsl:stylesheet

xsl:template

xsl:text
xsl:transform
xsl:value-of
xsl:variable
xsl:when
xsl:with-param
current
document
element-available
format-number
function-available
generate-id
key
system-property
unparsed-entity-url

XPATH Functions

boolean
ceiling
concat
contains
count
false
floor
id
lang
last
local-name
name
namespace-uri
normalize-space
not
number
position

round

starts-with

string

string-length

substring

substring-after

substring-before

sum

translate

true

XSL Example

In this example, I create two lists in XML and use the XSL style sheet to set the style display the way I want to set it. Also HTML is embedded in the style sheet which allows an actual ordered list to be created.

The XML File

```
<?xml version="1.0"?>
<!DOCTYPE LANGLIST SYSTEM "langlist.dtd">
<?xml-stylesheet type="text/xsl" href="xmlstyle.xsl"?>
<LANGLIST>
   <TITLE>List of Items Important to Markup Languages</TITLE>
   <TITLE1>Languages</TITLE1>
      <LIST1>
      <LANGUAGES>SGML</LANGUAGES>
      <LANGUAGES>XML</LANGUAGES>
      <LANGUAGES>HTML</LANGUAGES>
      </LIST1>
   <TITLE2>Other Support</TITLE2>
      <LIST2>
      <OTHER>DTD</OTHER>
      <OTHER>DSSSL</OTHER>
      <OTHER>Style Sheets</OTHER>
     </LIST2>
</LANGLIST>
```

The DTD File

```
<!ELEMENT LANGLIST ANY>
<!ENTITY % Shape "(rect|circle|poly|default)">
<!ELEMENT TITLE  (#PCDATA)>
<!ELEMENT TITLE1  (#PCDATA)>
<!ELEMENT TITLE2  (#PCDATA)>
<!ELEMENT LIST1 (LANGUAGES)+>
<!ELEMENT LIST2 (OTHER)+>
<!ELEMENT LANGUAGES  (#PCDATA)>
<!ATTLIST LANGUAGES
   type %Shape; #IMPLIED>
```

```
    <!ELEMENT OTHER  (#PCDATA)>
    <!ATTLIST OTHER
       type (disc|square|circle) #IMPLIED>
```

This DTD file employs an ENTITY, just for demonstration purposes. An entity, in this case is similar to a variable with a value that is set to a string value. In this case the entity name is "Shape" and the string value is "(rect|circle|poly|default)". This value is used to set the attribute list (ATTLIST) for the LANGUAGES and OTHER elements, although these attributes are not used in this example.

The XML Style File

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

        <xsl:template match="/">
                <xsl:apply-templates select="LANGLIST/TITLE" />
                <xsl:apply-templates select="LANGLIST/TITLE1" />
                <xsl:apply-templates select="LANGLIST/LIST1" />
                <xsl:apply-templates select="LANGLIST/TITLE2" />
                <xsl:apply-templates select="LANGLIST/LIST2" />
        </xsl:template>

        <xsl:template match="TITLE">
                <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#008000'; font-weight: '600'; font-size: '22'; margin-top:
'12pt'; text-align: 'center'">
                <xsl:value-of />
                </SPAN>
                <BR/>
        </xsl:template>

        <xsl:template match="TITLE1">
                <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#000080'; font-weight: '400'; font-size: '20'; margin-top:
'12pt'">
                <xsl:value-of />
                </SPAN>
                <BR/>
        </xsl:template>

        <xsl:template match="LIST1">
                <UL style="display: 'list-item'; list-style-image: url
```

```
('bullet8.gif'); font-family: 'arial'; color: '#000000'; font-weight:
'400';  margin-left: '15pt'; margin-top: '12pt'; font-size: '18'">
                <xsl:for-each select="LANGUAGES">
                        <LI style="display: 'list-item'; list-style-
type: 'square'; list-style-image: url('bullet8.gif'); font-family:
'arial'; color: '#ff0000'; font-weight: '300';  margin-left: '15pt';
margin-top: '12pt'; font-size: '16'">
                        <xsl:value-of />
                        </LI>
                </xsl:for-each>
                </UL>
        </xsl:template>

        <xsl:template match="TITLE2">
                <SPAN STYLE="display: 'block'; font-family: 'arial';
color: '#000080'; font-weight: '400'; font-size: '20'; margin-top:
'12pt'">
                <xsl:value-of />
                </SPAN>
                <BR/>
        </xsl:template>

        <xsl:template match="LIST2">
                <UL style="display: 'list-item'; list-style-image: url
('bullet8.gif'); font-family: 'arial'; color: '#000000'; font-weight:
'400';  margin-left: '15pt'; margin-top: '12pt'; font-size: '18'">
                <xsl:for-each select="OTHER">
                        <LI style="display: 'list-item'; list-style-
type: 'square'; list-style-image: url('bullet8.gif'); font-family:
'arial'; color: '#0000ff'; font-weight: '200';  margin-left: '15pt';
margin-top: '12pt'; font-size: '14'">
                        <xsl:value-of "."/>
                        </LI>
                </xsl:for-each>
                </UL>
        </xsl:template>

</xsl:stylesheet>
```

In the section above between the <xsl:template match="/"> tag and </xsl:template> tag, the order of the "apply-templates" statements is very important since this determines the order they are displayed in. Also note that the text <xsl:value-of "."/> is used to refer to the XML object currently being processed in order to display the value of that object.

Click Here to see the LANGLIST.XML file (USE your browser's BACK button to return).

XML DOM

Warning! This page is not complete! The reader should refer to the DOM documentation at the World Wide Web Consortium web site.

The Document Object Model (DOM) is a programming model that represents parts of an XML document as objects. Each object is thereby accessible by referring to the object and subobjects with periods between them. Each object supports specific attributes and methods which allows dynamic content using XML. The DOM is cross platform compatible. It allows scripts and programs to access and update the content and style of elements in a document. The DOM presents documents as a hierarchy of nodes.

Node Types

- Leaf - Can not have anything below them.
- Normal - Nodes may have children nodes.

Node Types	May Contain
Document	Document type, Element, Processing Instruction, Comment
Document Fragment	Entity reference, Element, Processing Instruction, Comment, Text, CDATA Section
Document Type	-
Entity Reference	Entity reference, Element, Processing Instruction, Comment, Text, CDATA Section
Element	Entity reference, Element, Processing Instruction, Comment, Text, CDATA Section
Attr	Entity reference, Textn
Processing Instructioon	-
Comment	-
Text	-
CDATA Section	-
Entity	Entity reference, Element, Processing Instruction, Comment, Text, CDATA Section
Notation	-

Node List - To handle lists of nodes. Named Node Max - For unorded sets of nodes. APIs in DOM are generally interfaces, not classes.

Type Definitions

DOMString - UTF-16 sequence of character units. also bound to the "String" type.

DOM Exceptions

DOM exceptions are defined by "exception DOMException". They are:

- DOMSTRING_SIZE_ERR
- HIERARCHY_REQUEST_ERR
- INDEX_SIZE_ERR
- INUSE_ATTRIBUTE_ERR
- INVALID_CHARACTER_ERR
- NOT_FOUND_ERR
- NOT_SUPPORTED_ERR
- NO_DATA_ALLOWED_ERR
- NO_MODIFICATION_ALLOWED_ERR
- WRONG_DOCUMENT_ERR

DOMImplementation Interface

Methods:

- hasFeature

DocumentFragment Interface

Document Interface

DOM Objects and Subobjects

- Core object
  - Array
  - Boolean
  - Date
  - Function
  - Math
  - Number
  - Object
  - String

    - RegExp
- Document
    - Link
    - Area
    - Anchor
    - Image
    - Applet
    - Layer
- Window
    - frame
    - Location
    - History
    - Screen
- Form
    - Hidden
    - Text
    - Textarea
    - Password
    - FuleUpload
    - Button
    - Submit
    - Reset
    - Radio
    - Checkbox
    - Select
    - Option

XML OSD

OSD describes XML components and their relationships with other components. This technology is supported and used by Microsoft. It is used to install and register software components on client computers. Open Software Description (OSD) elements used in an OSD specification and some attributes include:

- ABSTRACT - Used to provide a summary of information about the software.
- CODEBASE - Defines the URL where the installation files are located.
- DEPENDENCY
- DISKSIZE
- IMPLEMENTATION - Describes the hardware of operating system the package may be installed on.
- IMPLTYPE
- LANGUAGE
- LICENSE
- OS
- OSVERSION
- PROCESSOR
- SOFTPKG - This element begins and ends an OSD file. Some attributes are listed below.
    - name
    - version
- TITLE - The title of the package.
- VM

Distribution units are packaged with Compressed Cabinent Files (CAB) and INF or OSD files. The INF or OSD files contain the installation information. The above elements are used to provide the installation information for the software to be installed on the client machine. It specifies various limitations or ability of the package to be installed on various operating system platforms. The OSD file also includes the title and version of the package.

Tools

FrontPage, HoTMetaL The most widely used parser for SGML and XML is nsgmls, part of the SP suite written by James Clark

XML Terms

- CDF - Channel Definition Format
- DOM - Document Object Model is a group of objects that represent parts of an XML document.
- DSO - Data Source Object programming model allows scripts or data binding to be used to display an XML document from HTML.
- DSSSL -Document Style and Semantics Specification Language is a standard used to support SGML document transformation and display.
- DTD - Document Type Definition
- EBNF - Extended Backus-Naur Format is a standard format used to declare programming languages used by XML.
- OSD - Open Software Description
- RDF - Resource Description Framework is used to describe resources and their relationships to each other.
- SAX - Simple API for XML
- SGML - Standardized Generalized Markup Language is an international standard, ISO 8879:1986.
- XLL - Extensible linking Language

Credits

This document was produced for the Computer Technology Documentation Project and the latest version is available at http://www.comptechdoc.org/independent/xml/guide/.

Document:

The CTDP XML Guide Version 0.2.1

Author:

Mark Allen

Those who contributed by submitting comments:

- **Thore Harald Hoye**