There are four central problems in data management: capture, storage, retrieval, and exchange of data. The purpose of this book is to address XML, a technology for managing data exchange. The foundational XML chapters in this book are structured by a ‘data model’ approach. The first chapter introduces the reader to the XML document, XML schema, and XML stylesheet with a single entity example. Subsequent chapters expand upon the XML basics with multiple-entity examples and a one-to-one relationship, a one-to-many relationship, or a many-to-many relationship.

XML is a tool used for data exchange. Data exchange has long been an issue in information technology, but the Internet has elevated its importance. Electronic data interchange (EDI), the traditional data exchange standard for large organizations, is giving way to XML, which is likely to become the data exchange standard for all organizations, irrespective of size.

EDI supports the electronic exchange of standard business documents and is currently the major data format for electronic commerce. A structured format is used to exchange common business documents (e.g., invoices and shipping orders) between trading partners. In contrast to the free form of e-mail messages, EDI supports the exchange of repetitive, routine business transactions. Standards mean that routine electronic transactions can be concise and precise. The main standard used in the United States and Canada is known as X.12, and the major international standard is UN/EDIFACT. Firms adhering to the same standard can share data electronically.

The Internet is a global network potentially accessible by nearly every firm, with communication costs typically less than those of traditional EDI. Consequently, the Internet has become the electronic transport path of choice between trading partners. The simplest approach is to use the Internet as a means of transporting EDI documents. But because EDI was developed in the 1960s, another approach is to reexamine the technology of data exchange. A result of this rethinking is XML, but before considering XML we need to learn about SGML, the parent of XML.

SGML

For a typical U.S. firm, it is estimated that document management consumes up to 15 percent of its revenue, nearly 25 percent of its labour costs, and anywhere between 10 and 60 percent of an office worker’s time. The Standard Generalized Markup Language (SGML) is designed to reduce the cost and increase the efficiency of document management.

A markup language embeds information about a document within the document’s text. In the following example, the markup tags indicate that the text contains details of a city. Note also that the city’s name, state, and population are identified by specific tags. Thus, the reader—a person or a computer—is left in no doubt as to the meaning of both and XML. The Standard Generalized Markup Language (SGML) is designed to reduce the cost and increase the efficiency of document management.

SGML has three major advantages for data management:

- **Reuse:** Information can be created once and reused many times.
- **Flexibility:** SGML documents can be published in any format. The same content can be printed, presented on the Web, or delivered with a text synthesis. Because SGML is content-oriented, presentation decisions can be delayed until the output format is decided.
- **Revision:** SGML supports revision and version control. With content version control, a firm can readily track the changes in documents.

A short section of SGML demonstrates clearly the features and strength of SGML (see Exhibit 2). The tags surrounding a chunk of text describe its meaning and thus support presentation and retrieval. For example, the pair of tags <airline> and </airline> surrounding “Delta” identify the airline making the flight.
Delta flight 22 flies from Atlanta to Paris leaving 5:40pm and arriving 8:10am

<table>
<thead>
<tr>
<th>Airline</th>
<th>Flight</th>
<th>Origin</th>
<th>Destination</th>
<th>Departure</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta</td>
<td>22</td>
<td>Atlanta</td>
<td>Paris</td>
<td>5:40pm</td>
<td>8:10am</td>
</tr>
</tbody>
</table>

If the data are stored in HTML format and rendered on a Web site (as in Exhibit 3), then the meaning of the data has to be inferred by the reader. This is generally quite easy for humans, but impossible for machines. Furthermore, the presentation format is fixed and can only be altered by rewriting the HTML. If you are not familiar with HTML, you should read the WikiBooks chapter on XHTML, an extension of HTML, before reading the next chapter.

Exhibit 3: HTML rendering example

| Delta flight 22 flies from Atlanta to Paris leaving 5:40pm and arriving 8:10am |

Meaning and presentation should be independent, and this is an important reason why SGML is more powerful than HTML.

SGML is a markup language that defines the structure of documents and is preferred to HTML as it can be transformed into a variety of media.

XML

Many computer systems contain data in incompatible formats. A time-consuming challenge is to exchange data between such systems. XML is a generic data storage format that comes bundled with a number of tools and technologies that should make it easier to exchange specific XML 'applications' between incompatible systems. Since XML is open and generic, it is expected that as time progresses, more and more organizations and people will jump onto the XML bandwagon, both developers and data users. This should make XML the ultimate viable technology for certain types of data exchange.

XML is used not only for exchanging information, but also for publishing Web pages. XML's very strict syntax allows for smaller and faster Web browsers and as such is well suited for use with Personal Digital Assistants (PDAs) and cellphones. Web browsers that interpret HTML documents, on the other hand, are bloated with programming code to compensate for HTML's not so strict coding.

The types of data generally well suited for encoding as XML are those where field lengths are unknown and unpredictable and where field contents are predominantly textual.

An XML schema allows for the exchange of information in a standardized structure. A schema defines custom markup tags that can contain attributes to describe the content that is enclosed by these tags. Information from the tagged data in the XML document can be extracted using an application called a "parser", and with the use of an XML stylesheet the data can be formatted for a Web page.

XML's power lies in the combination of custom markup tags and content in a defined XML document. The purpose of eXtensible Markup Language (XML) is to make information self-describing. Based on SGML, XML is designed to support electronic commerce. The definition of XML, completed in early 1998 by the World Wide Web Consortium (W3C), describes it as a meta language — a language to generate languages. XML should steadily replace HTML on many Web sites because of some key advantages. The major differences between XML and HTML are captured in the following table.

Exhibit 4: XML vs HTML

<table>
<thead>
<tr>
<th>XML</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information content</td>
<td>Information presentation</td>
</tr>
<tr>
<td>Extensible set of tags</td>
<td>Fixed set of tags</td>
</tr>
<tr>
<td>Data exchange language</td>
<td>Data presentation language</td>
</tr>
<tr>
<td>Greater hypertext linking</td>
<td>Limited hypertext linking</td>
</tr>
</tbody>
</table>

The eXtensible in XML means that a new data exchange language can be created by defining its structure and tags. For example, the OpenGIS Consortium (http://www.opengis.org/) designed a Geography Markup Language (GML) to facilitate the electronic exchange of geographic information. Similarly, the Open Tourism Consortium (http://www.opentourism.org) is working on the definition of TourML to support exchange of tourism information. The insurance industry uses data corresponding to the XML based standard ACORD (http://www.acord.org) for electronic data exchange. Another good example of XML in action is NewsML™ (http://www.newsml.org).

In this text we will cover all the features of XML, but at this point let us introduce a few of the key features.

Applications of XML:

Before we start learning more about how an XML document is structured, let us point out what XML can be used for. The four major implementations of XML are:

Publication: Database content can be converted into XML and afterwards into HTML by using an XSLT stylesheet. Making use of this technique, complex websites as well as print media like PDF files can be generated. Information no longer has to be stored in different formats (i.e. RTF, DOC, PDF, HTML). Content can be stored in the neutral XML format and then, using appropriate layout style sheets and transformations, brochures, websites, or datalists can be generated (See more in Chapter 17.)
An example of the capability of XML and XSLT can be found at http://www.emimusic.de. This website contains approximately 20,000 pages with profiles of the artists, their products and the titles of the songs. These pages are generated using a XSLT script. Based on the script used it will also be possible to create a catalog in PDF format. Please see below for more details.

**Interaction:** XML can be used for accessing and changing data interactively. This man<–>machine communication usually happens via a web browser (see Chapter 12).

**Integration:** Using XML, homogenous and heterogenous applications can be integrated. In this case, XML is used to describe data, interfaces, and protocols. This machine-machine communication helps integrate relational databases (i.e. by importing and exporting different formats).

**Transaction:** XML helps to process transactions in applications like online marketplaces, supply chain management, and e-procurement systems.

### Key features of XML

- Elements have both an opening and a closing tag
- Elements follow a strict hierarchy, with documents containing only one root element
- Elements cannot overlap other elements
- Element names must obey XML naming conventions
- XML is case sensitive

XML will improve the efficiency of data exchange in several important ways, which include:

- **write once and format many times:** Once an XML file is created it can be presented in multiple ways by applying different XML stylesheets. For instance, the information might be displayed on a web page or printed in a book.
- **hardware and software independence:** XML files are standard text files, which means they can be read by any application.
- **write once and exchange many times:** Once an industry agrees on a XML standard for data exchange, data can be readily exchanged between all members using that standard.
- **Faster and more precise web searching:** When the meaning of information can be determined by a computer (by reading the tags), web searching will be enhanced. For example, if you are looking for a specific book title, it is far more efficient for a computer to search for text between the pair of tags `<booktitle>` and `</booktitle>` than search an entire file looking for the title. Furthermore, spurious results should be eliminated.
- **data validation** XML allows data validation using XSD or DTD which is a contractual agreement between two interacting parties.

### 10 reasons to use XML

1. XML is a widely accepted open standard.
2. XML allows to clearly separate content from form (appearance).
3. XML is text-oriented.
4. XML is extensible.
5. XML is self-describing.
6. XML is universal; meaning internationalization is no problem.
7. XML is independent from platforms and programming languages.
8. XML provides a robust and durable format for information storage.
9. XML is easily transformable.
10. XML is a future-oriented technology.

### The major XML elements

The major XML elements are:

- **XML document:** An XML file containing XML code.
- **XML schema:** An XML file that describes the structure of a document and its tags.
- **XML stylesheet:** An XML file containing formatting instructions for an XML file.

In the next few chapters you will learn how to create and use each of these elements of XML.

### Creating a markup file

Any text editor can be used to create a markup file (e.g. an HTML file). In this book, we use the text editor within NetBeans, an open source Integrated Development Environment (IDE) for Java, because NetBeans supports editing and validation of XML files. Before proceeding, you should download and install NetBeans from http://www.NetBeans.org/.

The examples in this book use NetBeans to illustrate proper XML code. For an alternative to NetBeans, see Exchanger XML Lite

### Case Studies in XML Implementation

**XML at United Parcel Service (UPS)**

“UPS is a service company and it is all about scale and speed,” says Geoff Chalmers, Project Leader at UPS eSolutions Department. In 2003, UPS had $33.5 billion annual revenue and 357,000 employees worldwide. Six percent of the United States’ Gross Domestic Product (GDP) on any given day is in the UPS system.

UPS uses technology extensively. The Information Systems department employs 4,000 people. The company's web site has 166 different country home pages and is supported by 44 applications.

UPS delivers around 13 million packages every day, and customers can track these shipments via the UPS Web site, which receives around 200 million hits daily. Nineteen of the applications within ups.com are XML OnLine Tool (Web services) applications.

UPS’s online tools are developed specifically to be integrated with customers’ applications. This makes the customer’s task simpler, easier, and faster. UPS verified the importance of simplicity and speed, via “CampusShip (http://www.ups.com/content/us/en/bussol/offering/technology/automated_shipping/pbn.html),” a product that has been one of the UPS’s most successful in the last 10 years. UPS CampusShip® is a Web-based, UPS-hosted shipping system. Using an Internet connection,
employees can ship their own packages and letters from any desktop, while management maintains overall control of shipping activities. UPS CampusShip® allows simultaneous shipper autonomy and managerial cost-control within the organization. This product has been successful because no installation or software maintenance is required and it is quick to implement. XML Online Tools enabled cheap and fast evolution of CampusShip®.

UPS favors XML especially because it is agnostic; platform and language independent. These features make XML very flexible and powerful. It is also decoupled and scalable. XML has enabled UPS to target a broader market and reduce customer interaction, and thus the cost of customer service. Another positive feature of XML is that it is backward compatible. The adoption of XML has reduced maintenance, implementation, and usage costs significantly within UPS.

However these advantages don’t come without a price. “XML is inefficient in so many ways” says Chalmers. XML unfortunately takes more CPU and bandwidth than the other technologies. Yet bandwidth and CPU are cheap and getting cheaper everyday, so this is a gradually disappearing problem.

Nevertheless, Chalmers also thinks that XML doesn’t work well in databases. He says that it is too wordy and it is an exchange medium rather than a database medium. There were some early attempts to tightly integrate XML and databases. Because databases do supply structure and identification to data as does XML, the value-add of XML-database integration is limited to applying hierarchical structure. On the other hand, if data is to be stored as a blob, then XML makes sense. Another problem that he points out about XML is that business rules cannot be expressed in XML schemas.

Finally, raw XML programming and debugging can be challenging. Therefore, UPS’s enterprise customers are starting to explore the code generators and embedded facilities to be found in .NET and BEA. However, hand coding by experienced in-house engineers is a must for the high availability, scalability, and performance that UPS requires for the UPS OnLine Tools.

**XML at EMI Music**

**How is it used?**

EMI Music Germany GmbH & Co. KG, a famous German record label, displays information about the artists it is affiliated with on its website (http://www.emi.de/). Visitors are able to explore all their audio or video productions. The whole website consists of nearly 20,000 pages that contain information about artists and their products (CD, DVD, LP). Everything is properly linked and systematically grouped.

After all, there is data to be provided for every artist, albums, samples, pictures, descriptions or article codes. The site is updated on a daily basis and is subject to change by a web editor whenever it’s necessary. Now this is a fairly complex and large amount of data to be handled.

This is where XML comes into play. The data, which is stored in a database, has been transformed into XML code. Now an XSLT stylesheet converts this data into HTML code, which can be easily read by any web browser (e.g. Internet Explorer or Firefox).

**What's the benefit?**

The advantage of XML is that the programming effort is considerably lower as compared to other formats. This is because XML lies at the point of intersection of XSLT and HTML.

It’s also no problem for the web editor to update the website. Using XML makes it easy for the person in charge to deal with this large amount of data.

**Going beyond...** On the basis of the XML scripts thus far produced by EMI Music, the company could easily produce a PDF-formatted catalog or design i-Mode pages for the current mobile phone generation. Thanks to XML, this can be done with little extra effort.

**A brief history of XML**

In the late 60s Charles Goldfarb, Raymond Lorie and Edward Mosher all working for IBM started to develop GML (Generalized Markup Language), a text formatting language. The language was successfully applied for internal documentation procedures. As it used to be common, the document editing was performed in the batch-mode. GenCode, another procedure to define generic formatting codes for the typesetting systems of various software producers, was developed by the GCA (Graphic Communications Association) at about the same time. Both of these technologies, GML syntactically and GenCode semantically, served as basis for the development of SGML (Standard Generalized Markup Language). The process of standardization started at the U.S. Standardization institute ANSI in the early 80s and in 1986 SGML finally passed as ISO standard ISO2879:1986.

SGML is reckoned to be a complex and comprehensive language (the specification extends 500 pages). However, the success of HTML (Hyper Text Markup Language) proved that the concepts of SGML were appropriate. SGML-based HTML was developed by Tim Berners-Lee in Geneva, in the early 90s in order to illustrate and link documents in the Internet. Meanwhile, HTML developed as the most successful format for all electronic documents. The Internet was originally designed as a space for human-human and human-machine communication but lately machine-machine communication has gained tremendous importance, putting a completely new challenge on the computer languages used.

HTML is a descriptive language for the presentation of documents. The main focus is on the presentation, meaning that an HTML-document mixes the presented data and its formatting instruction. A human being may recognize the displayed semantic by means of the presentation and the context meaning; a machine or (better-said) software is unable to.

In 1996 a team under the guidance of Jos Bosak attending the W3C-consortium was established to make SGML web-suible. The result was a 30-page specification, which received in February 1998 the status of a “W3C-recommendation” and was named “Extensible Markup Language (XML)”. The most important goals developing XML were:

- XML should be compatible with SGML
- XML should be easy to use in the Internet
- The number of optional characteristics should be minimized
- XML-documents should be easy to generate and human-readable
- XML should be supported by a variety of application
- It should be easy to write programs for XML
- XML should be put into practice on time

In the terminology of markup languages, a description formulated in XML is called a XML-document, albeit the content has nothing to do with text processing.

**Why is this book not an XML document?**

If you have accepted the ideas presented in this chapter, the question is very pertinent. The simple answer is that we have been unable to find the technology to support the creation of an open text book in XML. We need several pieces of technology

- An XML language for describing a book. DocBook (http://www.docbook.org/) is such a language, but the structure of a book is quite complex, and DocBook
(reflecting this complexity) cannot be quickly mastered

- A Wiki that works with a language such as DocBook
- A XML stylesheet that converts XML into HTML for displaying the book's content

There is a project to create WikiMI (http://wiki.wikiml.org/index.php?WikiMI) (Wiki MarkupLanguage), and this might be used at some point.

References

Initiating author Richard T. Watson (mailto:rwatson@terry.uga.edu), University of Georgia

---

### Learning objectives

- introduce XML documents, schemas, and stylesheets
- describe and create an XML document
- describe and create an XML schema
- describe and create an XML stylesheet

### Introduction

In this chapter, we start to practice working with XML using XML documents, schemas, and stylesheets. An XML document organizes data and information in a structured, hierarchical format. An XML schema provides standards and rules for the structure of a given XML document. An XML schema also enables data transfer.

An XSL (XML stylesheet) allows unique presentations of the material found within an XML document.

In the first chapter, Introduction to XML, you learned what XML is, why it is useful, and how it is used. So, now you want to create your very own XML documents. In this chapter, we will show you the basic components used to create an XML document. This chapter is the foundation for all subsequent chapters—it is a little lengthy, but don't be intimidated. We will take you through the fundamentals of XML documents.

This chapter is divided into three parts:

- XML Document
- XML Schema
- XML Stylesheets (XSL)

As you learned in the previous chapter, the XML Schema and StyleSheet are essentially specialized XML Documents. Within each of these three parts we will examine the layout and components required to create the document. There are links at the end of the XML document, schema, and stylesheet sections that show you how to create the documents using an XML editor. At the bottom of the page there is a link to Exercises for this chapter and a link to the Answers.

The first thing you will need before starting to create XML documents is a problem—something you want to solve by using XML to store and share data or information. You need some entity you can collect information about and then access in a variety of formats. So, we created one for you.

To develop an XML document and schema, start with a data model depicting the reality of the actual data that is exchanged. Once a high fidelity model has been created, the data model can be readily converted to an XML document and schema. In this chapter, we start with a very simple situation and in successive chapters extend the complexity to teach you more features of XML.

Our starting point is a single entity, CITY, which is shown in the following figure. While our focus is on this single entity, to map CITY to an XML schema, we need to have an entity that contains CITY. In this case, we have created TOURGUIDE. Think of a TOURGUIDE as containing many cities, and in this case TOURGUIDE has no attributes nor an identifier. It is just a container for data about cities.

**Exhibit 1: Data model - Tourguide**
XML document

An XML document is a file containing XML code and syntax. XML documents have an .xml file extension.

We will examine the features & components of the XML document.

- Prologue (XML Declaration)
- Elements
- Attributes
- Rules to follow
- Well-formed & Valid XML documents

Below is a sample XML document using our TourGuide model. We will refer to it as we describe the parts of an XML document.

Exhibit 2: XML document for city entity

<?xml version="1.0" encoding="UTF-8"?>
<tourGuide xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='city.xsd'>
  <city>
    <cityName>Belmopan</cityName>
    <adminUnit>Cayo</adminUnit>
    <country>Belize</country>
    <population>11100</population>
    <area>5</area>
    <elevation>130</elevation>
    <longitude>88.44</longitude>
    <latitude>17.27</latitude>
    <description>Belmopan is the capital of Belize</description>
    <history>Belmopan was established following the devastation of the former capital, Belize City, by Hurricane Hattie in 1965. High ground and open space influenced the choice and groundbreaking began in 1966. By 1979 most government offices and operations had already moved to the new location.</history>
  </city>
  <city>
    <cityName>Kuala Lumpur</cityName>
    <adminUnit>Derong</adminUnit>
    <country>Malaysia</country>
    <population>1448600</population>
    <area>243</area>
    <elevation>111</elevation>
    <longitude>101.71</longitude>
    <latitude>3.16</latitude>
    <description>Kuala Lumpur is the capital of Malaysia and the largest city in the nation</description>
    <history>The city was founded in 1857 by Chinese tin miners and preceded Klang. In 1885 the British government transferred their headquarters from Klang to Kuala Lumpur, and in 1896 it became the capital of Malaysia.</history>
  </city>
  <city>
    <cityName>Winnipeg</cityName>
    <adminUnit>St. Boniface</adminUnit>
    <country>Canada</country>
    <population>618512</population>
    <area>124</area>
    <elevation>40</elevation>
    <longitude>97.26</longitude>
    <latitude>49.54</latitude>
    <description>Winnipeg has two seasons. Winter and Construction.</description>
    <history>The city was founded by people at the Forks (Fort Garry) trading in pelts with the Hudson Bay Company. Ironically, the Bay was bought by America.</history>
  </city>
</tourGuide>

Prologue (XML declaration)

The XML document starts off with the prologue. The prologue informs both a reader and the computer of certain specifications that make the document XML compliant. The first line is the XML declaration (and the only line in this basic XML document).

Exhibit 3: XML document - prologue

<?xml version="1.0" encoding="UTF-8"?>
xml = this is an XML document
version="1.0" = the XML version (XML 1.0 is the W3C-recommended version)
encoding="UTF-8" = the character encoding used in the document - UTF 8 corresponds to 8-bit encoded Unicode characters (i.e. the standard way to encode international documents) - Unicode (http://www.unicode.org/) provides a unique number for every character.
Another potential attribute of the XML declaration:
standalone="yes" = the dependency of the document (yes' indicates that the document does not require another document to complete content)

Elements

The majority of what you see in the XML document consists of XML elements. Elements are identified by their tags that open with < or </ and close with > or />. The start tag looks like this: <element attribute="value">, with a left angle bracket (<) followed by the element type name, optional attributes, and finally a right angle bracket (>). The end tag looks like this: </element>, similar to the start tag, but with a slash (/) between the left angle bracket and the element type name, and no attributes.

When there's nothing between a start tag and an end tag, XML allows you to combine them into an empty element tag, which can include everything a start tag can: <img src="Belize.gif"/>. This one tag must be closed with a slash and right angle bracket (/>), so that it can be distinguished from a start tag.

The XML document is designed around a major theme, an umbrella concept covering all other items and subjects; this theme is analyzed to determine its component parts, creating categories and subcategories. The major theme and its component parts are described by elements. In our sample XML document, 'tourGuide' is the major theme; 'city' is a category; 'population' is a subcategory of 'city'; and the hierarchy may be carried even further: 'males' and 'females' could be subcategories of 'population'. Elements follow several rules of syntax that will be described in the Rules to Follow section.

We left out the attributes within the <tourGuide> start tag — that part will be explained in the XML Schema section.

Exhibit 4: Elements of the city entity XML document

```
<tourGuide>
  <city>
    <cityName>Belmopan</cityName>
    <adminUnit>Cayo</adminUnit>
    <country>Belize</country>
    <population>11100</population>
    <area>5</area>
    <elevation>130</elevation>
    <longitude>88.44</longitude>
    <latitude>17.27</latitude>
    <description>Belmopan is the capital of Belize</description>
    <history>Belmopan was established following the devastation of the former capital, Belize City, by Hurricane Hattie in 1965. High ground and open space influenced the choice and ground-breaking began in 1966. By 1970 most government offices and operations had already moved to the new location.</history>
  </city>
</tourGuide>
```

Element hierarchy

- **root element** - This is the XML document's major theme element. Every document must have exactly one and only one root element. All other elements are contained within this one root element. The root element follows the XML declaration. In our example, <tourGuide> is the root element.

- **parent element** - This is any element that contains other elements, the child elements. In our example, <city> is a parent element.

- **child element** - This is any element that is contained within another element, the parent element. In our example, <population> is a child element of <city>.

- **sibling element** - These are elements that share the same parent element. In our example, <cityName>, <adminUnit>, <country>, <population>, <area>, <elevation>, <longitude>, <latitude>, <description>, and <history> are all sibling elements.

Attributes

Attributes aid in modifying the content of a given element by providing additional or required information. They are contained within the element's opening tag. In our sample XML document code we could have taken advantage of attributes to specify the unit of measure used to determine the area and the elevation (it could be feet, yards, meters, kilometers, etc.); in this case, we could have called the attribute 'measureUnit' and defined it within the opening tag of 'area' and 'elevation'.

```
@adminUnit class="state">Cayo</adminUnit>
```

The above attribute example can also be written as:

1. using child elements

```
@adminUnit
  <class>state</class>
  <name>Cayo</name>
</adminUnit>
```

```
@adminUnit
```

2. using attribute

```
@adminUnit class="state">Cayo</adminUnit>
```
2. using an empty element

```xml
<adminUnit class="region" name="Selangor" />
```

Attributes can be used to:
- provide more information that is not defined in the data
- define a characteristic of the element (size, color, style)
- ensure the inclusion of information about an element in all instances

Attributes can, however, be a bit more difficult to manipulate and they have some constraints. Consider using a child element if you need more freedom.

**Rules to follow**

These rules are designed to aid the computer reading your XML document.

- The first line of an XML document must be the XML declaration (the prologue).
- The main theme of the XML document is established in the root element and all other elements must be contained within the opening and closing tags of this root element.
- Every element must have an opening tag and a closing tag - **no exceptions**
  (e.g. `<element>data stuff</element>`).
- Tags must be nested in a particular order => the parent element's opening and closing tags must contain all of its child elements' tags; in this way, you close first the tag that was opened last:

```xml
<parentElement>
  <childElement1>data</childElement1>
  <childElement2>
    <subChildElementA>data</subChildElementA>
    <subChildElementB>data</subChildElementB>
  </childElement2>
  <childElement3>data</childElement3>
</parentElement>
```
- Attribute values should have quotation marks around them and no spaces.
- Empty tags or empty elements must have a space and a slash (/) at the end of the tag.
- Comments in the XML language begin with `"<!--"` and end with `"-->".

**XML Element Naming Convention**

Any name can be used but the idea is to make names meaningful to those who might read the document.

- XML elements may only start with either a letter or an underscore character.
- The name must not start with the string "xml" which is reserved for the XML specification.
- The name may not contain spaces.
- The ":" should not be used in element names because it is reserved to be used for namespaces (This will be covered in more detail in a later chapter).
- The name may contain a mixture of letters, numbers, or other characters.

XML documents often have a corresponding database. The database will contain fields which correspond to elements in the XML document. A good practice is to use the naming rules of your database for the elements in the XML documents.

**DTD (Document Type Definition) Validation - Simple Example**

**Simple Internal DTD**

```xml
<?xml version="1.0"?>
<!DOCTYPE cdCollection [
  <!ELEMENT cdCollection (cd)>  
  <!ELEMENT cd (title, artist, year)>  
  <!ELEMENT title (#PCDATA)>  
  <!ELEMENT artist (#PCDATA)>  
  <!ELEMENT year (#PCDATA)>  
]>  
<cdCollection>  
  <cd>
```
Every element that will be used MUST be included in the DTD. Don’t forget to include the root element, even though you have already specified it at the beginning of the DTD. You must specify it again, in an <!ELEMENT> tag. <!ELEMENT cdCollection (cd)> The root element, <cdCollection>, contains all the other elements of the document, but only one direct child element: <cd>. Therefore, you need to specify the child element (only direct child elements need to be specified) in the parentheses. <!ELEMENT cd (title, artist, year)> With this line, we define the <cd> element. Note that this element contains the child elements <title>, <artist>, and <year>. These are spelled out in a particular order. This order must be followed when creating the XML document. If you change the order of the elements (with this particular DTD), the document won’t validate. <!ELEMENT title (PCDATA)> The remaining three tags, <title>, <artist>, and <year> don’t actually contain other tags. They do however contain some text that needs to be parsed. You may remember from an earlier lecture that this data is called Parsed Character Data, or #PCDATA. Therefore, #PCDATA is specified in the parentheses. So this simple DTD outlines exactly what you see here in the XML file. Nothing can be added or taken away, as long as we stick to this DTD. The only thing you can change is the #PCDATA text part between the tags.

Adding complexity

There may be times when you will want to put more than just character data, or more than just child elements into a particular element. This is referred to as mixed content. For example, let’s say you want to be able to put character data OR a child element, such as the <b> tag into a <description> element:

```xml
<!ELEMENT description (PCDATA | b | i)>
```

This particular arrangement allows us to use PCDATA, the <b> tag, or the <i> tag all at once. One particular caveat though, is that if you are going to mix PCDATA and other elements, the grouping must be followed by the asterisk (*) suffix. This declaration allows us to now add the following to the XML document (after defining the individual elements of course)

```xml
<cd>
  <artist>Gwen Stefani</artist>
  <year>2004</year>
  <genre>pop</genre>
  <description>This is a great album from former <nowiki><i>No Doubt</i></nowiki> singer <b>Gwen Stefani</b>. </description>
</cd>
```

With attributes this is done a little differently than with elements. Please see following example:

```xml
<cd remaster_date="1992">
  <title>Dark Side of the Moon</title>
  <artist>Pink Floyd</artist>
  <year>1973</year>
</cd>
```

In order for this to validate, it must be specified in the DTD. Attribute content models are specified with:

```xml
<!ATTLIST element_name attribute_name attribute_type default_value>
```

Let’s use this to validate our CD example:

```xml
<!ATTLIST cd remaster_date CDATA #IMPLIED>
```

Choices

```xml
<!ATTLIST person gender (male|female) "female">
```

Grouping Attributes for an Element

If a particular element is to have many different attributes, group them together like so:

```xml
<!ATTLIST car horn CDATA #REQUIRED
 seat CDATA #REQUIRED
 steeringwheel CDATA #REQUIRED
 price CDATA #IMPLIED>
```

Adding STATIC validation, for items that must have a certain value

```xml
<!ATTLIST classlist classNumber CDATA #IMPLIED
 building (WINNIPEG_DCE|WINNIPEG_MAIN) "WINNIPEG_MAIN"
 originalDeveloper CDATA #FIXED "Khal Shariff">
```

Suffixes

So what happens with our last example with the CD collection, when we want to add more CDs? With the current DTD, we cannot add any more CDs without getting an error. Try it and see. When you specify a child element (or elements) the way we did, only one of each child element can be used. Not very suitable for a CD collection is it? We can use something called suffixes to add functionality to the <!ELEMENT> tag. Suffixes are added to the end of the specified child element(s). There are 3 main suffixes that can be used:

- ( No suffix ): Only 1 child can be used.
- (+): One or more elements can be used.
Validating for multiple children with a DTD

So in the case of our CD collection XML file, we can add more CDs to the list by adding a + suffix:

```xml
<!ELEMENT cd_collection(cd+)>  
```

Using more internal formatting tags

Bold tags, B's for example are also defined in the DTD as elements, that are optional like thus:

```xml
<!ELEMENT notes (#PCDATA | b | i)*>  
<!ELEMENT b (#PCDATA)*>  
<!ELEMENT i (#PCDATA)*>  
```

Case Study on BMEcat

One of the first major national projects for the use of XML as a B2B exchange format was initiated by the federal association for material management, purchasing and logistics (BME) in cooperation with leading German companies, e.g. Bayer, BMW, SAP and Siemens. They all created a standard for the exchange of product catalogues. This project was named BMEcat (http://www.bmecat.org/English/index.asp?). The result of this initiative is a DTD collection for the description of product catalogues and related transactions (new catalogue, updating of product data and updating of prices).

Companies operating in the electronic commerce (suppliers, purchasing companies and market places) exchange increasingly large amounts of data. They quickly reach their limits here by the variety of data exchange formats. The BMEcat solution creates a basis for a straightforward transfer of catalogue data from various data formats. This lays the foundation to bringing forward the goods traffic through the Internet in Germany. The use of the BMEcat reduces the costs for all parties as standard interfaces can be used.

The XML-based standard BMEcat was successfully implemented in many projects. Nowadays a variety of companies applies BMEcat and use it for the exchange of their product catalogs in this established standard.

A BMEcat catalogue (Version 1.2) consists of the following main elements:

**CATALOG** This element contains the essential information of a shopping catalog, e.g. language version and validity. BMEcat expects exactly one language per catalog.

**SUPPLIER** This element includes identification and address of the catalog suppliers. BMEcat expects exactly one supplier per catalog.

**BUYER** This element contains the name and address of the catalogue recipient. BMEcat expects no more than one recipient per catalog.

**AGREEMENT** This element contains one or more framework agreement IDs associated with the appropriate validity period. BMEcat expects all prices of a catalogue belonging to the contract mentioned above.

**CLASSIFICATION SYSTEM** This element allows the full transfer of one or more classification systems, including feature definitions and key words.

**CATALOG GROUP SYSTEM** This element originates from version 1.0. It is mainly used for the transfer of tree-structures which facilitate the navigation of a user in the target system (Browser).

**ARTICLE** (since 2005 PRODUCT) This element represents a product. It contains a set of standard attributes.

**ARTICLE PRICE** (since 2005 PRODUCT PRICE) This element represents a price. The support of different pricing models is very powerful in comparison with other exchange formats. Season prices, country prices, different currencies and different validity periods, etc. will be supported.

**ARTICLE FEATURE** (since 2005 PRODUCT FEATURE) This element allows the transfer of characteristic values. You can either record predefined group characteristics or individual product characteristics.

**VARIANT** This element allows listing of product variants, without having to duplicate them. However, the variations of BMEcat only apply to individual changes in value, leading to a change of Article ID. Otherwise there can’t exist any dependences on other attributes (especially at prices).

**MIME** This element includes any number of additional documents such as product images, data sheets, or websites.

**ARTICLE REFERENCE** (since 2005 REFERENCE PRODUCT) This element allows cross-referencing between articles within a catalogue as well as between catalogues. These references may used restrictedly for mapping product bundles.

**USER DEFINED EXTENSION** This element enables transportation of data at the outside the BMEcat standards. The transmitter and receiver have to be coordinated.
You can find a typical BMEcat file here. (http://de.wikipedia.org/wiki/BMECat)

**ONLINE Validator**

http://www.stg.brown.edu/service/xmlvalid/

**Well-formed and valid XML**

**Well-formed XML** - An XML document that correctly abides by the rules of XML syntax.

**Valid XML** - An XML document that adheres to the rules of an XML schema (which we will discuss shortly). To be valid an XML document must first be well-formed.

A Valid XML Document must be Well-formed. But, a Well-formed XML Document might not be valid - in other words, a well-formed XML document, that meets the criteria for XML syntax, might not meet the criteria for the XML schema, and will therefore be invalid.

For example, think of the situation where your XML document contains the following (for this schema):

```
<city>
  <cityName>Boston</cityName>
  <country>United States</country>
  <adminUnit>Massachusetts</adminUnit>
</city>
```

Notice that the elements do not appear in the correct sequence according to the schema (cityName, adminUnit, country). The XML document can be validated (using validation software) against its declared schema – the validation software would then catch the out of sequence error.

**Using an XML Editor**

Check chapter XML Editor for instructions on how to start an XML editor. Once you have followed the steps to get started you can copy the code in the sample XML document and paste it into the XML editor. Then check your results. Is the XML document well-formed? Is the XML document valid? (you will need to have copied and pasted the schema in order to validate - we will look at schemas next)

**XML schema**

An XML schema is an XML document. XML schemas have an .xsd file extension.

An XML schema is used to govern the structure and content of an XML document by providing a template for XML documents to follow in order to be valid. It is a guide for how to structure your XML document as well as indicating your XML document's components (elements and attributes - and their relationships). An XML editor will examine an XML document to ensure that it conforms to the specifications of the XML schema it is written against - to ensure it is valid.

XML schemas engender confidence in data transfer. With schemas, the receiver of data can feel confident that the data conforms to expectations. The sender and the receiver have a mutual understanding of what the data represent.

Because an XML schema is an XML document, you use the same language - standard XML markup syntax - with elements and attributes specific to schemas.

A schema defines:

- the structure of the document
- the elements
- the attributes
- the child elements
- the number of child elements
- the order of elements
- the names and contents of all elements
- the data type for each element


**Schema reference**

This is the part of the XML Document that references an XML Schema:

```
<tourGuide
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="city.xsd">
```

This is the part we left out when we described the root element in the basic XML document from the previous section. The additional attributes of the root element <tourGuide> reference the XML schema (it is the schemaLocation attribute).
Schema document

Below is a sample XML schema using our TourGuide model. We will refer to it as we describe the parts of an XML schema.

**Exhibit 6: XML schema document for city entity**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">
  <xsd:element name="tourGuide">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="cityDetails">
    <xsd:sequence>
      <xsd:element name="cityName" type="xsd:string"/>
      <xsd:element name="adminUnit" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="population" type="xsd:integer"/>
      <xsd:element name="area" type="xsd:integer"/>
      <xsd:element name="elevation" type="xsd:integer"/>
      <xsd:element name="longitude" type="xsd:decimal"/>
      <xsd:element name="latitude" type="xsd:decimal"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="history" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Prolog

Remember that the XML schema is essentially an XML document and therefore must begin with the prolog, which in the case of a schema includes:

- the XML declaration
- the schema element declaration

The XML declaration:

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

The schema element declaration:

```xml
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"/>
```

The schema element is similar to a root element - it contains all other elements in the schema.

Attributes of the schema element include:

- `xmlns` - XML NameSpace - the URL for the site that describes the XML elements and data types used in the schema.
- You can find more about namespaces here => Namespace.
- `xmlns:xsd` - All the elements and attributes with the 'xsd' prefix adhere to the vocabulary designated in the given namespace.
- `elementFormDefault` - elements from the target namespace are either required or not required to be qualified with the namespace prefix. This is mostly useful when more than one namespace is referenced. In this case, 'elementFormDefault' must be qualified, because you must indicate which namespace you are using for each element. If you are referencing only one namespace, then 'elementFormDefault' can be unqualified. Perhaps, using qualified as the default is most prudent, this way you do not accidentally forget to indicate which namespace you are referencing.

**Element declarations**

Define the elements in the schema.

Include:

- the element name
- the element data type (optional)

Basic element declaration format: `<xsd:element name="name" type="type"/>
```

**Simple type**

declares elements that:

- do NOT have Child Elements
- do NOT have Attributes
Default Value
If an element is not assigned a value then the default value is assigned.

example: `<xsd:element name="cityName" type="xsd:string" />`

Fixed Value
An attribute that is defined as fixed must be empty or contained the specified fixed value. No other values are allowed.

example: `<xsd:element name="description" type="xsd:string" default="really cool place to visit!" />`

Complex type
declares elements that:
  - can have Child Elements
  - can have Attributes

examples:
1. The root element 'tourGuide' contains a child 'city'. This is shown here:

   **Nameless complex type**

   ```xml
   <xsd:element name="tourGuide">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
       </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
   ```

   Occurrence Indicators:
   - minOccurs = the minimum number of times an element can occur (here it is 1 time)
   - maxOccurs = the maximum number of times an element can occur (here it is an unlimited number of times, 'unbounded')

2. The parent element 'city' contains many child elements: 'cityName', 'adminUnit', 'country', 'population', etc. Why does this complex element set not start with the line: `<xsd:element name="city" type="cityDetails"/>`? The element 'city' was already defined above within the complex element 'tourGuide' and it was given the type, 'cityDetails'. This data type, 'cityDetails', is utilized here in identifying the sequence of child elements for the parent element 'city'.

   **Named Complex Type** - and therefore can be reused in other parts of the schema

   ```xml
   <xsd:complexType name="cityDetails">
     <xsd:sequence>
       <xsd:element name="cityName" type="xsd:string"/>
       <xsd:element name="adminUnit" type="xsd:string"/>
       <xsd:element name="country" type="xsd:string"/>
       <xsd:element name="population" type="xsd:integer"/>
       <xsd:element name="area" type="xsd:integer"/>
       <xsd:element name="elevation" type="xsd:integer"/>
       <xsd:element name="longitude" type="xsd:decimal"/>
       <xsd:element name="latitude" type="xsd:decimal"/>
       <xsd:element name="description" type="xsd:string"/>
       <xsd:element name="history" type="xsd:string"/>
     </xsd:sequence>
   </xsd:complexType>
   ```

   The `<xsd:sequence>` tag indicates that the child elements must appear in the order, the sequence, specified here.

   Compare the sample XML Schema and the sample XML Document - try to observe patterns in the code and how the XML Schema sets up the XML Document.

3. Elements that have attributes are also designated as complex type.
   a. this XML Document line: `<adminUnit class="state" name="Cayo" />` would be defined in the XML Schema as:

   ```xml
   <xsd:element name="adminUnit">
     <xsd:complexType>
       <xsd:attribute name="class" type="xsd:string" />
       <xsd:attribute name="name" type="xsd:string" />
     </xsd:complexType>
   </xsd:element>
   ```

   b. this XML Document line: `<adminUnit class="state">Cayo</adminUnit>` would be defined in the XML Schema as:

   ```xml
   <xsd:element name="adminUnit">
     <xsd:complexType>
       <xsd:simpleContent>
         <xsd:extension base="xsd:string">
           <xsd:attribute name="class" type="xsd:string" />
         </xsd:extension>
       </xsd:simpleContent>
     </xsd:complexType>
   </xsd:element>
   ```

   **Attribute declarations**
Attribute declarations are used in complex type definitions. We saw some attribute declarations in the third example of the Complex Type Element.

```xml
<xs:attribute name="class" type="xsd:string"/>
```

**Data type declarations**

These are contained within element and attribute declarations as `type=""`.

**Common XML Schema Data Types**

XML schema has a lot of built-in data types. The most common types are:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>a string of characters</td>
</tr>
<tr>
<td>decimal</td>
<td>a decimal number</td>
</tr>
<tr>
<td>integer</td>
<td>an integer</td>
</tr>
<tr>
<td>boolean</td>
<td>the values true or false or 1 or 0</td>
</tr>
<tr>
<td>date</td>
<td>a date, the date pattern can be specified such as YYYY-MM-DD</td>
</tr>
<tr>
<td>time</td>
<td>a time of day, the time pattern can be specified such as HH:MM:SS</td>
</tr>
<tr>
<td>dateTime</td>
<td>a date and time combination</td>
</tr>
<tr>
<td>anyURI</td>
<td>if the element will contain a URL</td>
</tr>
</tbody>
</table>

For an entire list of built-in simple data types see [http://www.w3.org/TR/xmlschema-2/#built-in-datatypes](http://www.w3.org/TR/xmlschema-2/#built-in-datatypes)

**Using an XML Editor => XML Editor**

This link will take you to instructions on how to start an XML editor. Once you have followed the steps to get started you can copy the code in the sample XML schema document and paste it into the XML editor. Then check your results. Is the XML schema well-formed? Is the XML schema valid?

**XML stylesheet (XSL)**

An XML Stylesheet is an XML Document. XML Stylesheets have an `.xsl` file extension.

The eXtensible StyleSheets Language (XSL) provides a means to transform and format the contents of an XML document for display. Since an XML document does not contain tags a browser understands, such as HTML tags, browsers cannot present the data without a stylesheet that contains the presentation information. By separating the data and the presentation logic, XSL allows people to view the data according to their different needs and preferences.

The XSL Transformation Language (XSLT) is used to transform an XML document from one form to another, such as creating an HTML document to be viewed in a browser. An XSLT stylesheet consists of a set of formatting instructions that dictate how the contents of an XML document will be displayed in a browser, with much the same effect as Cascading StyleSheets (CSS) do for HTML. Multiple views of the same data can be created using different stylesheets. The output of a stylesheet is not restricted to a browser.

During the transformation process, XSLT analyzes the XML document and converts it into a node tree – a hierarchical representation of the entire XML document. Each node represents a piece of the XML document, such as an element, attribute or some text content. The XSL stylesheet contains predefined “templates” that contain instructions on what to do with the nodes. XSLT will use the match attribute to relate XML element nodes to the templates, and transform them into the resulting document.

**Exhibit 7: XML stylesheet document for city entity**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Tour Guide</title>
      </head>
      <body>
        <h2>Cities</h2>
        <xsl:apply-templates select="tourGuide"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="tourGuide">
    <xsl:for-each select="city">
      <xsl:value-of select="continentName"/>
      <br/>
      <xsl:value-of select="cityName"/>
      <br/>
      <xsl:text>Population: </xsl:text>
      <xsl:value-of select="format-number(population, "##,###,###")"/>
      <br/>
      <xsl:value-of select="country"/>
      <br/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```
The output of the city.xsl stylesheet in Table 2-3 will look like the following:

<table>
<thead>
<tr>
<th>Cities</th>
<th>Europe</th>
<th>Madrid</th>
<th>Population: 3,128,600</th>
<th>Spain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cities</td>
<td>Asia</td>
<td>Shanghai</td>
<td>Population: 18,880,000</td>
<td>China</td>
</tr>
</tbody>
</table>

You will notice that the stylesheet consists of HTML to inform the media tool (a web browser) of the presentation design. If you do not already know HTML, this may seem a little confusing. Online resources such as the W3Schools tutorials can help with the basic understanding you will need =>(http://www.w3schools.com/html/default.asp).

Incorporated within the HTML is the XML that supplies the data, the information, contained within our XML document. The XML of the stylesheet indicates what information will be displayed and how. So, the HTML constructs a display and the XML plugs in values within that display. XSL is the tool that transforms the information into presentational form, but at the same time keeps the meaning of the data.

XML at Bertelsmann - a case study

The German Bertelsmann Inc. is a privately owned media conglomerate operating in 56 countries. It has interests in such businesses as TV broadcast (RTL), magazine (Gruner & Jahr), books (Random House) etc. In 2005 its 89,000 employees generated 18 billion € of revenue.

A major concern of such a diversified business is utilizing synergies. Management needs to make sure the Random House employees don’t spend time and money figuring out what RTL TV journalists already have come up with.

Thus knowledge management based on IT promises huge time savings. Consequently Bertelsmann in 2002 started a project called BeCom. BeCom’s purpose was to enable the different Bertelsmann businesses to use the same data for their different media applications. XML is crucial in this project, because it allows for separating data (document) from presentation (style sheet). Thus data can both be examined statistically and be modified to fit different media like TV and newspapers.

Statistical XML data management for example enables employees to benefit from CBR (Case Based Reasoning). CBR allows a Bertelsmann employee who searches for specific content to profit from previous search findings of other Bertelsmann employees, thus gaining info which is much more contextual than isolated research results only. Besides XML data management, Bertelsmann TV and Book units can apply this optimized data in their specific media using a variety of lay-out applications like 3B2 or QuarkXPress.

Prolog

- the XML declaration;
- the stylesheet declaration;
- the namespace declaration;
- the output document format.

The XML declaration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

The stylesheet & namespace declarations

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

- identifies the document as an XSL style sheet;
- identifies the version number;
- refers to the W3C XSL namespace - the URL for the site that describes the XML elements and data types used in the schema. You can find more about namespaces here => Namespace. Every time the xsl. prefix is used it references the given namespace.

The output document format

```xml
<xsl:output method="html"/>
```

This element designates the format of the output document and must be a child element of `<xsl:stylesheet>`

Templates
The `<xsl:template>` element is used to create templates that describe how to display elements and their content. Above, in the XSL introduction, we mentioned that XSL breaks up the XML document into nodes and works on individual nodes. This is done with templates. Each template within an XSL describes a single node. To identify which node a given template is describing, use the 'match' attribute. The value given to the 'match' attribute is called a pattern. Remember: (node tree – a hierarchical representation of the entire XML document. Each node represents a piece of the XML document, such as an element, attribute or some text content). Wherever there is branching in the node tree, there is a node. `<xsl:template>` defines the start of a template and contains rules to apply when a specified node is matched.

the match attribute

`<xsl:template match="/"/>

This template match attribute associates the XML document root (/), the whole branch of the XML source document, with the HTML document root. Contained within this template element is the typical HTML markup found at the beginning of any HTML document. This HTML is written to the output. The XSL looks for the root match and then outputs the HTML, which the browser understands.

`<xsl:template match="tourGuide">

This template match attribute associates the element 'tourGuide' with the display rules described within this element.

Elements

Elements specific to XSL:

<table>
<thead>
<tr>
<th>XSL Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:text&gt;</code></td>
<td>Prints the actual text found between this element's tags</td>
</tr>
<tr>
<td><code>&lt;xsl:value-of&gt;</code></td>
<td>This element is used with a 'select' attribute to look up the value of the node selected and plug it into the output.</td>
</tr>
<tr>
<td><code>&lt;xsl:for-each&gt;</code></td>
<td>This element is used with a 'select' attribute to handle elements that repeat by looping through all the nodes in the selected node set.</td>
</tr>
<tr>
<td><code>&lt;xsl:apply-templates&gt;</code></td>
<td>This element will apply a template to a node or nodes. If it uses a 'select' attribute then the template will be applied only to the selected child node(s) and can specify the order of child nodes. If no 'select' attribute is used then the template will be applied to the current node and all its child nodes as well as text nodes.</td>
</tr>
</tbody>
</table>

For more XSL elements => [http://www.w3schools.com/xsl/xsl_w3celementref.asp](http://www.w3schools.com/xsl/xsl_w3celementref.asp).

Language-Specific Validation and Transformation Methods

PHP Methods of XML Dom Validation


Browser Methods

Place this line of code in your .xml document after the XML declaration (prologue).

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

PHP XML Production

```php
<?php
$xmlData = "";
mysqli_connect('localhost','root','')
or die('Failed to connect to the DBMS');
// make connection to database
mysqli_select_db('issd')
or die('Failed to open the requested database');
$result = mysqli_query('SELECT first_name FROM students')
or die('Query to like get the records failed');
if (mysqli_num_rows($result) <= 0) {die('');}
$xmlString = '<classlist>
';
$xmlString .='<student>' .
while ($row = mysqli_fetch_array($result)) {
$xmlString .= '<firstName>' . $row['first_name'] . '</firstName>' .
$xmlString .= '<lastName>' . $row['last_name'] . '</lastName>' .
_xml:template match="tourGuide">

try {
$xmlfile = "classList.xml"; // any file
if ($fh = fopen($xmlfile, "w")) or die("can't open file") { // create filehandler
fwrite($fh, $xmlString); // write the data into the file
```
PHP Methods of XSLT Transformation

This one is good for PHP5 and wampserver (latest). Please ensure that *xsl* is NOT commented out in the php.ini file.

```php
// load the xsl source
$xml = new DOMDocument;
$xml->load('tourguide.xml');
$xsl = new DOMDocument;
$xsl->load('tourguide.xsl');
// Configure the transformer
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl);
// attach the xsl rules
echo $proc->transformToXML($xml); // ALL DONE!
```

Example 1, Using within PHP itself (use phpInfo() function to check XSLT extension; enable if needed) This example might produce XHTML. Please note it could produce anything defined by the XSL.

```php
<?php
$xhtmlOutput = xslt_create();
$args = array();
$params = array('foo' => 'bar');
$theResult = xslt_process($xsltproc, 'theContentSource.xml', 'theTransformationSource.xsl', null, $args, $params);
xslt_free($xhtmlOutput); // free that memory
// echo theResult or save it to a file or continue processing (perhaps instructions)
?>
```

Example 2:

```php
if (PHP_VERSION >= 5) {
    // Emulate the old xslt library functions
    function xslt_create() {
        return new XsltProcessor();
    }
    function xslt_process($xsltproc, $xml_arg, $xsl_arg, $xslcontainer = null, $args = null, $params = null) {
        // Start with preparing the arguments
        $xml_arg = str_replace('arg:', '', $xml_arg);
        $xsl_arg = str_replace('arg:', '', $xsl_arg);
        // Create instances of the DomDocument class
        $xml = new DOMDocument;
        // Load the xml document and the xsl template
        $xml->loadXML($args['$xml_arg']);
        $xsl = new DOMDocument;
        // Load the xsl template
        $xsltproc->importStyleSheet($xsl);
        // Set parameters when defined
        if ($params) {
            foreach ($params as $param => $value) {
                $xsltproc->setParameter('', $param, $value);
            }
        }
        // Start the transformation
        $processed = $xsltproc->transformToXML($xml);
        // Put the result in a file when specified
        if ($xslcontainer) {
            return @file_put_contents($xslcontainer, $processed);
        } else {
            return $processed;
        }
    }
    function xslt_free($xsltproc) {
        unset($xsltproc);
    }
    $arguments = array(
        '/_xml' => file_get_contents('xml_files/201945.xml'),
        '/_xsl' => file_get_contents('xml_files/convertToSql_new2.xsl')
    );
    $xsltproc = xslt_create();
    $xml = xslt_process($xsltproc, 'arg:/_xml', 'arg:/_xsl', null, $arguments);
    xslt_free($xsltproc);
}
?>
```

PHP file writing code

```php
$file = "testFile.xml"; // any file
if (fopen($myFile, "w") or die("can't open file"); // create filehandler
StringData = "<foo><bar><hello>"); // get a string ready to write
fwrite($fh, $StringData); // write the data into the file
StringData2 = "<tt><br>
<foo>";
```
**XML Colors**

For use in your stylesheet: these colors can be used for both background and font

http://www.w3schools.com/html/html_colors.asp

http://www.w3schools.com/html/html_colorsfull.asp

http://www.w3schools.com/html/html_colornames.asp

Using an XML Editor => XML Editor

This link will take you to instructions on how to start an XML editor. Once you have followed the steps to get started you can copy the code in the sample XML stylesheet document and paste it into the XML editor. Then check your results. Is the XML stylesheet well-formed?

**XML at Thomas Cook - a case study**

As the leading travel company and most widely recognized brands in the world, Thomas Cook works across the travel value chain - airlines, hotels, tour operators, travel and incoming agencies, providing its customers with the right product in all market segments across the globe. Employing over 11,000 staff, the Group has 33 tour operators, around 3,600 travel agencies, a fleet of 80 aircraft and a workforce numbering some 26,000. Thomas Cook operates throughout a network of 616 locations in Europe and overseas. The company is now the second largest travel group in Europe and the third largest in the world.

As Thomas Cook sells other companies’ products, ranging from packaged holidays to car hires, it needs to regularly change its online brochure. Before Thomas Cook started using XML, it put information into HTML format, and would take up to six weeks to get an online brochure up and running online. XML helps do this job in about three days. This helps provide all of Thomas Cook’s current and potential customers and its various agencies in different geographical locations with updated information, instead of having to wait six weeks for new information to be released.

XML allows Thomas Cook to put content information into a single database, which can be re-used as many times as required. "We did not want to keep having to re-do the same content, we wanted the ability to switch it on immediately," said Gwyn Williams, who is content manager at Thomascook.com. "This has brought internal benefits such as being able to re-deploy staff into more value added areas." Thomascook.com currently holds 65,000 pages of brochure and travel guide information and an online magazine in XML format.

Thomas Cook started using XML at a relatively early stage. As Thomas Cook has a large database, the early use of XML will stand it in good stead. At some point, the databases will have to be incorporated into XML, and it is reported that XML databases are quicker than conventional databases, giving Thomas Cook a slight competitive advantage against those who do not use XML.

Thomas Cook has found that this can lead to substantial cost reductions as well as consistency of information across all channels. By implementing a central content management system to facilitate brochure production and web publications, they have centralized the production, maintenance and distribution of content across their brands and channels.

**Summary**

From the previous chapter *Introduction to XML*, you have learned the need for data exchange and the usefulness of XML in data exchange. In this chapter, you have learned more about the three major XML files: the XML document, the XML schema, and the XML stylesheet. You learned the correct documentation required for each type of file. You learned basic rules of syntax applicable for all XML documents. You learned how to integrate the three types of XML documents. And you learned the definition and distinction between a *well-formed* document and a *valid* document. By following the XML Editor links, you were able to see the results of the sample code and learn how to use an XML Editor.

Below are Exercises and Answers for further practice. Good Luck!

**Definitions**

XML

SGML

Dan Connolly

RSS

XML Declaration

parent

child

sibling

element

attribute

--Well-Formed XML---

--PCDATA---
Exercises

Exercise 1.

a) Using "tourguide" above as a good example, create an XML document whose root is "classlist". This CLASSLIST is created from a starting point of single entity, STUDENT. Any number of students contain elements: firstname, lastname, emailaddress.

Answers

Introduction

In a one-to-many relationship, one object can reference several instances of another. A model is mapped into a schema whereby each data model entity becomes a complex element type. Each data model attribute becomes a simple element type, and the one-to-many relationship is recorded as a sequence.

Exhibit 1: Data model for 1:m relationship

In the previous chapter, we introduced a simple XML schema, XML document, and an XML stylesheet for a single entity data model. We now include more features of each of the key aspects of XML.

Implementing a one-to-many relationship

There are three different techniques for implementing a one-to-many relationship:

Containment relationship: A structure is defined where one element is contained within another. The "contained" element ceases to exist when the "container" element is removed. For instance, where a city has many hotels, the hotels are "contained" in the city.

Intra-document relationships: In a case where you have one city with many hotels, rather than a city containing hotels, a hotel will have a "location in" relationship to a city. A city id is used as a reference on the hotel element. Therefore, rather than the hotels being contained in the city, they now just reference the city's id via the cityRef attribute. This is very similar to a foreign key in a relational database.
Inter-document relationships: The inter-document relationship is much like the intra-document relationship. It also uses the id and idRef attributes to assign an attribute to a parent attribute. The difference is that the inter-document relationship is used when tables, such as the city and hotel tables, might live in different filesystems or tablespaces.

Exhibit 2: Checklist for deciding what technique to use:

<table>
<thead>
<tr>
<th>Technique</th>
<th>Passing Data</th>
<th>Flexibility</th>
<th>Ease of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containment</td>
<td>Excellent</td>
<td>Fair</td>
<td>Excellent</td>
</tr>
<tr>
<td>Intra-Document</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Inter-Document</td>
<td>Fair</td>
<td>Excellent</td>
<td>Fair</td>
</tr>
</tbody>
</table>

XML schema

Some of the built-in data types for an XML schema were introduced in the previous chapter, but still, there are more that are very useful, such as anyURI, date, time, year, and month. In addition to the built-in data types, a custom data type can be defined by the schema designer to accept specific data input. As we have learned, data are defined in XML documents using markup tags defined in an XML schema. However, some elements might not have values. An empty element tag can be used to address this situation. An empty element tag (and any custom markup tag) can contain attributes that add additional information about the tag without adding extra text to the element. An example will be shown in the chapter, using attributes in an empty element tag.

Empty elements with attributes in XML document

Elements can have different content types depending on how each element is defined in the XML schema. The different types are element content, mixed content, simple content, and empty content. An XML element consists of everything from the start of the element tag to the close of that element tag.

- An element with element content is the root element - everything in between the opening and closing tags consists of elements only.

  Example: <tourGuide>
  
  </tourGuide>

- A mixed content element is one that has text and as well as other elements between its opening and closing tags.

  Example: <restaurant>My favorite restaurant is <restaurantName>Provino's Italian Restaurant</restaurantName>:
  
  <restaurant>

- A simple content element is one that contains only text between its opening and closing tags.
An empty content element, which is an empty element, is one that does not contain anything between its opening and closing tags (or the element tag is opened and ended with a single tag, by using / before the closing of the opening tag).

Example: 
<hotelPicture filename="pan_pacific.jpg" size="80" value="Image of Pan Pacific"/>

An empty element is useful when there is no need to specify its content or that the information describing the element is fixed. Two examples illustrated this concept. First, a picture element that references the source of an image with its attributes, but has no need in specifying text content. Second, the owner’s name is fixed for a company, thus it can specify the related information inside the owner tag using attributes. An attribute is meta-information, information that describes the content of the element.

European Central Bank's use of XML

<?xml version="1.0" encoding="UTF-8"?>
<gesmes:subject>
Reference rates
</gesmes:subject>
<gesmes:Sender>
<gesmes:name>
European Central Bank
</gesmes:name>
</gesmes:Sender>
<Cube>
<Cube time="2004-05-28">
<Cube currency="USD" rate="1.2246"/>
<Cube currency="JPY" rate="135.77"/>
<Cube currency="DKK" rate="7.4380"/>
<Cube currency="GBP" rate="0.66730"/>
<Cube currency="SEK" rate="9.1150"/>
<Cube currency="CHF" rate="1.5304"/>
<Cube currency="ISK" rate="87.72"/>
<Cube currency="NOK" rate="8.2120"/>
</Cube>
</Cube>
</gesmes:Envelope>

XML schema data types

Some of the commonly used data types, such as string, decimal, integer, and boolean, are introduced in chapter 2. The following are a few more data types that are useful.

Exhibit 3: Other data types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Format</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>YYYY</td>
<td>1999</td>
<td></td>
</tr>
<tr>
<td>month</td>
<td>YYYY-MM</td>
<td>1999-03</td>
<td>Month type is used when the day is irrelevant for the data element</td>
</tr>
<tr>
<td>time</td>
<td>hh:mm:ss.sss with optional time zone indicator</td>
<td>20:14:05</td>
<td>Z for UTC or one of −hh:mm or +hh:mm to indicate the difference from UTC. This time type is used when you want the content to represent a particular time of day that recurs every day, such as 4:15 pm.</td>
</tr>
<tr>
<td>date</td>
<td>YYYY-MM-DD</td>
<td>1999-03-14</td>
<td></td>
</tr>
<tr>
<td>anyURI</td>
<td>The domain name specified beginning with http://</td>
<td><a href="http://www.panpacific.com">http://www.panpacific.com</a></td>
<td></td>
</tr>
</tbody>
</table>

More data types

Besides the built-in data types, custom data types can be created as required. A custom data type can be a simple type or complex type. For simplicity, we create a custom data type that is a simple type, which means that the element does not contain other elements or attributes. It contains text only. The creation of a custom simple type starts from using a built-in simple type and applying it with restrictions, or facets, to limit the acceptable values of the tag. A custom simple type can be nameless or named. If the custom simple type is to be used only once, then it makes sense to not name it; thus, that custom type will only be used in where it is defined. Since a named custom type can be referenced (by its name), that custom type can be used wherever necessary.

A pattern can be used to specify exactly how the content of the element should look. For example, one might want to specify the format of a telephone number, a postal code, or a product code. By having a defined pattern for certain elements, the data exchanged will be uniform and the values will be consistent when stored in a database. A useful way to set patterns is through Regex, which will be discussed in later chapters.

Schema examples

The following is a schema that extends the schema introduced in the previous chapter to include a one-to-many relationship of city to hotels with two examples of custom data types.

Exhibit 1: Data model for 1:m relationship

1:m relationship - City Hotel

Important, this is a continuing example, so new code is added to the last chapter's example!

Containment example
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

<!--This will contain the City details-->
<xsd:complexType name="cityDetails">
    <xsd:sequence>
        <xsd:element name="cityID" type="xsd:ID"/>
        <xsd:element name="cityName" type="xsd:string"/>
        <xsd:element name="adminUnit" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<!--This will contain the Hotel details-->
<xsd:complexType name="hotelDetails">
    <xsd:sequence>
        <xsd:element name="hotelName" type="xsd:string"/>
        <xsd:element name="hotelPicture" type="xsd:anyURI"/>
        <xsd:element name="hotelRating" type="xsd:integer"/>
        <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0"/>
        <xsd:element name="hotelRating" type="xsd:integer"/>
        <xsd:element name="population" type="xsd:integer"/>
        <xsd:element name="area" type="xsd:string"/>
        <xsd:element name="latitude" type="xsd:decimal"/>
        <xsd:element name="longitude" type="xsd:decimal"/>
    </xsd:sequence>
</xsd:complexType>

<!--NOTE: Since postalCode, emailAddress, and websiteURL are not standard elements that must be provided, the minOccurs="0" indicates that they are optional-->

<!--This is a Named Custom SimpleType that is called from Hotel whenever someone types in an email address-->
<xsd:simpleType name="emailAddressType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\w+\s*\w*@{1}\w+\s*\w+\w+\w+.*\w*"/>
    </xsd:restriction>
</xsd:simpleType>

</xsd:schema>

Intra-document example

<xml version="1.0" encoding="UTF-8">
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

<!--Tour Guide-->
<xsd:element name="tourGuide">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

<!--This will contain the City details-->
</xsd:element>

</xsd:schema>
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <!-- This will contain the Hotel details.-->
  <xsd:complexType name="hotelDetails">
    <xsd:complexContent>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Hotel"/>
      </xsd:restriction>
      <xsd:sequence>
        <xsd:element name="hotelName" type="xsd:string"/>
        <xsd:element name="hotelRating" type="xsd:decimal"/>
        <xsd:element name="phone" type="xsd:string"/>
        <xsd:element name="emailAddressType" type="emailAddressType" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="history" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- The custom simple type, emailAddressType, defined in the xsd:complexType, is used as the type of the emailAddress element. -->
  <xsd:simpleType name="emailAddressType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\w+\W*\w*@\w+\W*\w+\..\w+\.*\w*"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- NOTE: Since postalCode, emailAddress, and websiteURL are not standard elements that must be provided, the minOccurs="0" indicates that they are optional. -->
  <xsd:complexType name="tourGuide">
    <xsd:complexContent>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="tourGuide"/>
      </xsd:restriction>
      <xsd:sequence>
        <xsd:element name="tourGuide2" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="cityDetails" type="xsd:IDRef" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="history" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="description" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="latitude" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="longitude" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="population" type="xsd:integer" maxOccurs="unbounded"/>
        <xsd:element name="area" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="elevation" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="areaRef" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="country" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="adminUnit" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityName" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="city" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityRef" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="continent" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="area" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="elevation" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="areaRef" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="country" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="adminUnit" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityName" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="city" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityRef" type="xsd:IDRef" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- This will contain the City details.-->
  <xsd:complexType name="cityDetails">
    <xsd:complexContent>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="city"/>
      </xsd:restriction>
      <xsd:sequence>
        <xsd:element name="cityID" type="xsd:ID"/>
        <xsd:element name="cityName" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="adminUnit" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="country" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="area" type="xsd:integer" maxOccurs="unbounded"/>
        <xsd:element name="areaRef" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="elevation" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="population" type="xsd:integer" maxOccurs="unbounded"/>
        <xsd:element name="area" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="elevation" type="xsd:decimal" maxOccurs="unbounded"/>
        <xsd:element name="areaRef" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="country" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="adminUnit" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityName" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="city" type="xsd:IDRef" maxOccurs="unbounded"/>
        <xsd:element name="cityRef" type="xsd:IDRef" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexContent>
  </xsd:complexType>
  <!-- This will contain the Hotel details.-->
  <xsd:simpleType name="continent">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Asia"/>
      <xsd:enumeration value="Africa"/>
      <xsd:enumeration value="Australia"/>
      <xsd:enumeration value="Europe"/>
      <xsd:enumeration value="North America"/>
      <xsd:enumeration value="South America"/>
      <xsd:enumeration value="Antarctica"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
  
  <!-- This is a Named Custom SimpleType that is called from Hotel whenever someone types in an emailAddress Element. It must be provided, the minOccurs="0" indicates that they are optional. -->
  <xsd:simpleType name="emailAddressType">
   
   <xsd:restriction base="xsd:string">
    
    <!--You can learn more about this pattern by reading the Regex section.--> 
    <xsd:pattern value="[^\r\n]+"/>
   
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="hotel">
   
   <xsd:sequence>
    
   </xsd:sequence>
  
  </xsd:complexType>

  <xsd:element name="hotelName" type="xsd:string"/>

  <xsd:element name="hotelPicture"/>

  <xsd:element name="streetAddress" type="xsd:string"/>

  <xsd:element name="postalCode" type="xsd:string" minOccurs="0"/>

  <xsd:element name="phone" type="xsd:string"/>

  <xsd:element name="emailAddress" type="emailAddressType" minOccurs="0"/>

  <!-- The custom simple type, emailAddressType, defined in the xsd:complexType, is used as the type of the emailAddress element. -->

  <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0"/>

  <xsd:element name="hotelRating" type="xsd:int"/>

  </xsd:sequence>

  <!--NOTE: Since postalCode, emailAddress, and websiteURL are not standard elements that must be provided, the minOccurs="0" indicates that they are optional. -->

  <!--This is a Named Custom SimpleType that is called from Hotel whenever someone types in an emailAddress Element. It must be provided, the minOccurs="0" indicates that they are optional. -->

  <xsd:element name="~filename" type="xsd:string"/>

  <xsd:element name="~size" type="xsd:int"/>

  <xsd:element name="~value" type="xsd:string"/>

  </xsd:sequence>

  <xsd:schema>

  </xsd:schema>

References to Chapter 2 - A single entity for steps in using NetBeans to create the above XML schema.

XML document

Attributes

- The valid element naming structure applies to attribute names as well
- In a given element, all attributes' names must be unique
- An attribute may not contain the symbol '" The character string '" can be used to represent it
- Each attribute must have a name and a value. (i.e. <hotelPicture filename="pan_pacific.jpg" />, filename is the name and pan_pacific.jpg is the value)
- If the assigned value itself contains a quoted string, the type of quotation marks must differ from those used to enclose the entire value. (For instance, if double quotes are used to enclose the whole value then use single quotes for the string: <name familiar="'Jack'">John Smith</name>)

The valid element naming structure applies to attribute names as well.
In a given element, all attributes' names must be unique.

An attribute may not contain the symbol '" The character string '" can be used to represent it.
Each attribute must have a name and a value. (i.e. <hotelPicture filename="pan_pacific.jpg" />, filename is the name and pan_pacific.jpg is the value)
If the assigned value itself contains a quoted string, the type of quotation marks must differ from those used to enclose the entire value. (For instance, if double quotes are used to enclose the whole value then use single quotes for the string: <name familiar="'Jack'">John Smith</name>)
<city>
  <cityName>Kuala Lumpur</cityName>
  <adminUnit>Selangor</adminUnit>
  <country>Malaysia</country>
  <continent>Asia</continent>
  <population>1448600</population>
  <area>243</area>
  <elevation>111</elevation>
  <longitude>101.71</longitude>
  <latitude>3.16</latitude>
  <description>
    Kuala Lumpur is the capital of Malaysia and is the largest city in the nation. The city was founded in 1857 by Chinese tin miners and superseded Klang. In 1880 the British government transferred their headquarters from Klang to Kuala Lumpur, and in 1896 it became the capital of Malaysia.
  </description>
</city>
Exercises

In order to learn more about the one-to-many relationship, exercises are provided.

Answers

In order to learn more about the one-to-many relationship, answers are provided to go with the exercises above.

### Related Topics
- Computer Science Home
- Library and Information Science Home
- Markup Languages

### Get Involved
- To do list
- Contributors list
- Contributing to Wikibooks

---

**Learning objectives**

- Create a schema for a data model containing a 1:1 relationship
- Place restrictions on elements or attributes in an XML schema
- Specify fixed or default values for an element in an XML schema

---

**Introduction**

In the previous chapter, some new features of XML schemas, documents, and stylesheets were introduced as well as how to model a one-to-many relationship. In this chapter, we will introduce the modeling of a one-to-one relationship in XML. We will also introduce more features of an XML schema.

### A one-to-one (1:1) relationship

The following diagram shows a one-to-one and a one-to-many relationship. The one-to-one relationship records each country as a single top destination.

![Diagram of a one-to-one relationship](image)

**Exhibit 4-1: Data model for a 1:1 relationship**

### XML schema

A one-to-one (1:1) relationship is represented in the data model in Exhibit 4-1. The addition of country and destination to the data model allows the 1:1 relationship named topDestination. A country has many different destinations, but only one top destination. The XML schema in Exhibit 4-2 shows how to represent a 1:1 relationship in an XML schema.

### XML schema example

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  
</xsd:schema>
```
New elements in schema

Let’s examine the new elements and attributes in the schema in Exhibit 4-2.

- Country is a complex type defined in City to represent the 1:M relationship between a country and its cities.
- Destination is a complex type defined in Country to represent the 1:M relationship between a country and its many destinations.
- topDestination is a complex type defined in Country to represent the 1:1 relationship between a country and its top destination.

Restrictions in schema

Placing restrictions on elements was introduced in the previous chapter, however, there are more potentially useful restrictions that can be placed on an element. Restrictions can be placed on elements and attributes that affect how the processor handles whitespace characters:

White space & length constraints

The whiteSpace constraint is set to “preserve”, which means that the XML processor will not remove any white space characters. Other useful restrictions include the following:

- Replace – the XML processor will replace all whitespace characters with spaces.
  ```xml
  <xsd:whiteSpace value="replace"/>
  ```

- Collapse – The processor will remove all whitespace characters.
  ```xml
  <xsd:whiteSpace value="collapse"/>
  ```

- Length, maxLength, minLength—the length of the element can be fixed or can have a predefined range.
  ```xml
  <xsd:length value="8"/>
  <xsd:minLength value="5"/>
  <xsd:maxLength value="8"/>
  ```

Order indicators

In addition to placing restrictions on elements, order indicators can be used to define in what order elements should occur.

All indicator
The `<all>` indicator specifies by default that the child elements can appear in any order and that each child element must occur once and only once:

```xml
c<xsd:element name="person">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

**Choice indicator**

The `<choice>` indicator specifies that either one child element or another can occur:

```xml
c<xsd:element name="person">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="employee" type="employee"/>
      <xsd:element name="visitor" type="visitor"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

**Sequence indicator**

The `<sequence>` indicator specifies that the child elements must appear in a specific order:

```xml
c<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

**XML document**

The XML document in Exhibit 4-3 shows how the new elements (country and destination) defined in the XML schema found in Exhibit 4-2 are used in an XML document. Note that the child elements of `<topDestination>` can appear in any order because of the `<xsd:all>` order indicator used in the schema.

```xml
<tourGuide xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="XMLSchema.xsd">
  <!-- Malaysia -->
  <country>
    <countryName>Malaysia</countryName>
    <population>22229040</population>
    <continent>Asia</continent>
    <topDestination>
      <destinationName>Pulau Langkawi</destinationName>
      <description>A popular duty-free island north of Penang.</description>
    </topDestination>
  </country>
  <!-- Belize -->
  <country>
    <countryName>Belize</countryName>
    <population>249183</population>
    <continent>South America</continent>
    <topDestination>
      <destinationName>San Pedro</destinationName>
      <description>San Pedro is an island off the coast of Belize</description>
    </topDestination>
  </country>
</tourGuide>
```

Exhibit 4-3: XML Document for a one-to-one relationship
Summary

Schema designers may place restrictions on the length of elements and on how the processor handles white space. Schema designers may also specify fixed or default values for an element. Order indicators can be used to specify the order in which elements must appear in an XML document.

Exercises

Answers

Introduction

In the previous chapters, you learned how to use XML to structure and format data based on one-to-one and one-to-many relationships. Because XML provides the means to model data using hierarchical parent-child relationships, the one-to-one and one-to-many relationships are relatively simple to represent in XML. However, this hierarchical parent-child structure is difficult to use to model the many-to-many relationship, a common relationship between entities in many situations.

In this chapter, we will explore the pros and cons of a few methods that are used to model a many-to-many relationship in XML; these methods offer compromises in overcoming the problems that arise when applying this relationship to XML. In particular, we will see examples of how to model the many-to-many relationship using two different methods, “Eliminate” and “ID/IDREF.” Additionally, in the XML stylesheet, we will learn how to implement the key function to display the data that was modeled using the “ID/IDREF” method.

Problems: many-to-many relationship

In XML, the parent-child relationship is most commonly used to represent a relationship. This can easily be applied to a one-to-one or one-to-many relationship. A many-to-many relationship is not supported directly by XML; the parent-child relationship will not work as each element may only have a single parent element. There are couple of possible solutions to get around this.

Solutions: many-to-many relationship

Eliminate

Create XML documents that eliminate the need for a many-to-many relationship

By limiting the extent of information that is conveyed, you can get around the need for a many-to-many relationship. Instead of trying to have one XML document encompass all of the information, separate the information where one document describes only one of the entities that participates in the many-to-many relationship. Using our tourGuide relationship for example, one way for us to accomplish this would be creating a separate XML document for each hotel. The relationship with amenity would ultimately then become a one-to-many. This method is more suitable for situations in which the scope of data exchange can be limited to subsets of data. However, using this method for more broadly scoped data exchange, you may repeat data several times, especially if there are many attributes. To avoid this redundancy, use the ID/IDREF method.

ID/IDREF

Represent the many-to-many relationship using unique identifiers

Although not the most user-friendly way to handle this problem, one way of getting around the many-to-many relationship is by creating keys that would uniquely identify each entity. To do this, an element with ID or IDREF attributes-types must be specified within the XML schema. To use a data modeling analogy, ID is similar
to the primary key, and IDREF is similar to the foreign key.

**Many-to-many relationship data model**

Exhibit 1: Data model for a m:m relationship

![Diagram of a many-to-many relationship data model]

The relationship reads, a hotel can have many amenities, and an amenity can exist at many hotels.

As you will notice, in order to represent a many-to-many relationship, two entities were added. The middle entity is necessary for the data model to represent an associative entity that stores data about the relationship between hotel and amenity. Using our Tour Guide example, "Amenity" was added to represent a list of possible amenities that a hotel can possess.

The following examples illustrate methods to represent a many-to-many relationship in XML.

**Eliminate: sample solution**

In this example, the many-to-many relationship has been converted to a one-to-many relationship.

**XML schema**

Exhibit 2: XML schema for "Eliminate" method

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="hotelGuide">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="hotel" type="hotelDetails" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="emailAddressType">
    <xs:restriction base="xs:string">
      <xs:pattern value="\w+\W*\w*@\w+\W*\.\w+.\w+.*\w*"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="hotelDetails">
    <xs:sequence>
      <xs:element name="hotelPicture"/>
      <xs:element name="hotelName" type="xs:string"/>
      <xs:element name="streetAddress" type="xs:string"/>
      <xs:element name="telephoneNumber" type="xs:string"/>
      <xs:element name="emailAddress" type="emailAddressType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="hotelRating" type="xs:integer" default="0"/>
      <xs:element name="lowerPrice" type="xs:positiveInteger"/>
      <xs:element name="upperPrice" type="xs:positiveInteger"/>
      <xs:element name="amenity" type="amenityValue" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="amenityValue">
    <xs:sequence>
      <xs:element name="amenityType" type="xs:string"/>
      <xs:element name="amenityOpenHour" type="xs:time"/>
      <xs:element name="amenityCloseHour" type="xs:time"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

**XML document**

Exhibit 3: XML document for "Eliminate" method

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<hotelGuide xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="amenity1.xsd">
  <hotel>
    <hotelPicture/>
    <hotelName>Narembeen Hotel</hotelName>
    <streetAddress>Churchill Street</streetAddress>
    <telephoneNumber>+61 (08) 9064 7272</telephoneNumber>
    <emailAddress>narempub@oz.com.au</emailAddress>
    <hotelRating>1</hotelRating>
    <lowerPrice>50</lowerPrice>
    <upperPrice>100</upperPrice>
    <amenity>
      <amenityType>Restaurant</amenityType>
      <amenityOpenHour>06:00:00</amenityOpenHour>
    </amenity>
  </hotel>
</hotelGuide>
```
ID/IDREF: sample solution

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.

To avoid redundancy, we create a separate element, "amenity," which is included at the top of the schema along with "hotel." Remember, the data types ID and IDREF are synonymous with the primary key and foreign key, respectively. For every foreign key (IDREF), there must be a matching primary key (ID). Note that the IDREF data type has to be an alphanumeric string.

The following example illustrates the ID/IDREF approach. Notice that the ID for the amenity pool is defined as "k1," and every hotel with a pool as an amenity

data type has to be an alphanumeric string.
Key function: XML stylesheet

In order to set up an XML stylesheet using the ID/IDREF method for a many-to-many relationship, the key function should be used. In the stylesheet, the <xsl:key> element specifies the index, which is used to return a node-set from the XML document.

A key consists of the following:
1. the node that has the key
2. the name of the key
3. the value of a key

The following XML stylesheet illustrates how to use the key function to present content that is structured in a many-to-many relationship.

XML stylesheet

Exhibit 6: XML stylesheet for "ID/IDREF" method
Expedia.de: XML and affiliate marketing

Expedia.de is the German subsidiary of expedia.com, the internet-based travel agency headquartered in Bellevue, Washington, USA. It offers its customers the booking of airline tickets, car rentals, vacation packages and various other attractions and services via its website and by phone. Its websites attract more than 70 million visitors each month. Currently expedia.com employs 4.600 employees serving customers in the United States, Canada, the UK, France, Germany, Italy, and Australia.

For marketing purposes expedia.de set up an affiliate marketing program. Affiliate marketing is a way to reach potential customers without any financial risk for the company intending to advertise (merchant). The merchant gives website owners, which are called affiliates, the opportunity to refer to the merchant page, offering commission-based monetary rewards as incentives. In the case of Expedia.de the affiliate partners receive a commission every time users from their websites book travel on expedia.de. So the affiliates can concentrate on selling and the merchant takes care of handling the transactions.

To ease the business of the affiliate partners – and of course to make the program more attractive – Expedia.de offers its partners a service called xmlAdEd. xmlAdEd is a service providing current product information on using XML. Affiliates using this service are able to request more than 8 million of travel offerings in XML format via HTTP-request. The data is updated several times a day. In the HTTP-request you can set certain parameters such as location, price, airport code, …

The use of XML in this case gives the affiliates several advantages:
- Efficient and flexible processing of the data because of separation of structure, content and style.
- Platform-independent processing of the data.
- Lossless conversion into other file formats.
- Easy integration in their websites.
- Possibility to create an own web shop in individual design

By providing their affiliates product information in XML, expedia.de not only eases the business of their partners, but also ensures that customers receive consistent, up-to-date information on their services.

Summary

When describing a many-to-many relationship in XML, there are a few solutions available for designers to use. In choosing how to represent the many-to-many relationship, the designer not only must consider the most efficient way to represent the information, but also the audience for which the document is intended and how the document will be used.

References

http://www.w3.org/TR/xslt#key

Exercises

Answers
Introduction

Recursive relationships are an interesting and more complex concept than the relationships you have seen in the previous chapters. A recursive relationship occurs when there is a relationship between an entity and itself. For example, a one-to-many recursive relationship occurs when an employee is the manager of other employees. The employee entity is related to itself, and there is a one-to-many relationship between one employee (the manager) and many other employees (the people who report to the manager). Because of the more complex nature of these relationships, we will need slightly more complex methods of mapping them to a schema and displaying them in a style sheet.

The one-to-one recursive relationship

Continuing with the tour guide model, we will develop a schema that shows cities that have hosted the Olympics and the previous host city. Since the previous host is another city and only one city can be the previous host this is a one to one recursive relationship.

```
<cityID>Atlanta</cityID>
<cityID>Sydney</cityID>
<cityID>Athens</cityID>
```

Exhibit 1: XML schema for Host City Entity

host.xsd (XML schema for a one-to-one recursive model)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:element name="cityType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cityID" type="xs:ID"/>
      <xs:element name="cityName" type="xs:string"/>
      <xs:element name="cityCountry" type="xs:string"/>
      <xs:element name="cityPop" type="xs:integer"/>
      <xs:element name="cityHostYr" type="xs:integer"/>
      <xs:element name="cityPreviousHost" type="xs:IDREF" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Exhibit 1: XML schema for Host City Entity

host.xml (XML document for a one-to-one recursive model)
The one-to-many recursive relationship

A hypothetical sports team is divided into squads with each squad having a captain. Every person on the team is a player, regardless of whether they are a squad captain. Since a squad captain is a player, this situation meets the definition of a recursive relationship—a squad captain is also a player and has a one-to-many relationship with the other players. This is a one-to-many recursive relationship because one captain has many players under him/her. See the example below for how to model the relationship.

**team.xsd (XML schema for a one-to-many recursive model)**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema' elementFormDefault='unqualified'>
  <xs:element name='team'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='player' type='playerType' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name='playerType'>
    <xs:sequence>
      <xs:element name='playerID' type='xsd:ID'/>
      <xs:element name='playerName' type='xsd:string'/>
      <xs:element name='playerCap' type='playerC' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name='playerC'>
    <xs:sequence>
      <xs:element name='memberOf' type='xsd:IDREF'/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

**team.xml (XML document for a one-to-many recursive model)**

```xml
<?xml version='1.0' encoding='UTF-8'?>
<team xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='Recursive1toMSchema.xsd'>
  <player>
    <playerID>c1</playerID>
    <playerName>Tommy Jones</playerName>
    <playerCap>
      <memberof>memberOf</memberof>
    </playerCap>
  </player>

  <player>
    <playerID>c2</playerID>
    <playerName>Eddie Thomas</playerName>
    <playerCap>
      <memberof>memberOf</memberof>
    </playerCap>
  </player>

  <player>
    <playerID>c3</playerID>
    <playerName>Sean McCombs</playerName>
  </player>

  <player>
    <playerID>c4</playerID>
    <playerName>Patrick O'Shea</playerName>
    <playerCap>
      <memberof>memberOf</memberof>
    </playerCap>
  </player>
</team>
```

**The many-to-many recursive relationship**

A more natural approach for most one-to-many recursive relationships is to use XML's hierarchical nature to directly represent the hierarchy. Consider Locations:

```xml
<location type='country'>
  <name>USA</name>
  <sub-locations>
    <location type='state'>
      <name>Ohio</name>
      <sub-locations>
        <location type='city'>
          <name>Akron</name>
        </location>
        <location type='city'>
          <name>Columbus</name>
        </location>
      </sub-locations>
    </location>
  </sub-locations>
</location>
```

The many-to-many recursive relationship...
Think you're getting a feel for recursive relationships yet? Well, there is still the third and final relationship to add to your repertoire — the many-to-many recursive. A common example of a many-to-many recursive relationship is when one item can be comprised of many items of the same data type as itself, and each of those sub-items may belong to another parent item of the same data type. Sound confusing? Let's look at the example of a product that can consist of a single item or multiple items (i.e., a packaged product). The example below describes tourist products that can be packaged together to create a new product.

**Product.xsd (XML schema for a many-to-many recursive model)**

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="products">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="product" type="prodType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="prodType">
    <xs:sequence>
      <xs:element name="prodID" type="xsd:ID"/>
      <xs:element name="prodName" type="xsd:string" minOccurs="0"/>
      <xs:element name="prodCost" type="xsd:decimal"/>
      <xs:element name="prodPrice" type="xsd:decimal"/>
      <xs:element name="components" type="componentsType" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="componentsType">
    <xs:sequence>
      <xs:element name="component" type="xsd:IDREF"/>
      <xs:element name="componentqty" type="xsd:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

*Exhibit 5: XML schema for Product Entity*

**Product.xml (XML document for a many-to-many recursive model)**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="product.xsd">
  <product>
    <prodID>p1000</prodID>
    <prodName>Animal photography kit</prodName>
    <prodPrice>725</prodPrice>
    <components>
      <component>p101</component>
      <componentqty>1</componentqty>
    </components>
  </product>
  <product>
    <prodID>p101</prodID>
    <prodName>Camera case</prodName>
    <prodCost>150</prodCost>
    <prodPrice>300</prodPrice>
    <components>
      <component>p1000</component>
      <componentqty>1</componentqty>
    </components>
  </product>
</products>
```

*Exhibit 6: XML Document for Product Entity*

**Summary**

When the child has the same type of data as its parent in a parent-child type data relationship, this is a sign of the existence of a recursive relationship. The xsd:ID and xsd:IDREF elements can be used in a schema to create primary key-foreign key values in an XML document.

**Exercises**

**Answers**

**External Links**

- XML Schema (http://blog.eukhost.com/2006/10/20/xml-schema/)

Previous Chapter  Recursive relationships  Next Chapter  DTD
Introduction

Data schemas are the foundation of all XML pages. They define objects, their relationships, their attributes, and the structure of the data model. Without them, XML documents would not exist. In this chapter, you will come to understand the purpose of XML data schemas, their intricate parts, and how to utilize them. Also, examples will be included for you to copy when creating your own data schema, making your job a lot easier. At the bottom of this Web page a whole Schema has been included, from which parts have been included in the different sections throughout this chapter. Refer to it if you would like to see how the whole Schema works as one.

Overview of Data Schemas

The data schema, all technicalities aside, is the data model with which all the XML information is conveyed. It has a hierarchy structure starting with a root element (to be explained later) and goes all the way down to cover even the most minute detail of the model with detailed steps in between. Data schemas have two main parts, the entities and their relationships. The entities contained in a data schema represent objects from the model. They have unique identifiers, attributes, and names for what kind of object they are. The relationships in the schema represent the relationships between the objects, simple enough. Relationships can be one to one, one to many, many to many, recursive, and any other kind you could find in a data model. Now we will begin to create our own data schema.

Starting your schema the right way

All schemas begin the same way, no matter what type of objects they represent. The first line in every Schema is this declaration:

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

Exhibit 1: XML Declaration

Exhibit 1 simply tells the browser or whatever file/program accessing this schema that it is an XML file and uses the encoding structure "UTF-8". You can copy this to use to start your own XML file. Next comes the Namespace declaration:

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified">
```

Exhibit 2: Namespace Declaration

Namespaces are basically dictionaries containing definitions of most of the coding in the schema. For example, when creating a schema, if you declare an object to be of type "String", the definition of the type "String" is contained in the Namespace along with all of its attributes. This is true for most of the code you write. If you have made or seen other schemas, most of the code is prefaced by "xsd:"

A good example is something like "xsd:sequence" or "xsd:complexType". sequence and complexType are both objects defined in the Namespace that has been linked to the prefix "xsd". In fact, you could theoretically name the default Namespace anything, as long as you referenced it the same way throughout the Schema. The most common Namespace which contains most of the XML objects is http://www.w3.org/2001/XMLSchema. Now onto Exhibit 2.

The first part lets any file/program know that this file is a schema. Pretty easy to understand. Like the XML declaration, this is universal to XML schemas and you can use it in yours. The second part is the actual Namespace declaration; xmlns stands for XML NameSpace. This defines the Schema's default Namespace and is usually the one given in the code. Again, I would recommend using this code to start your Schemas. The last part is difficult to understand, but here is a pretty detailed explanation. Using "unqualified" is most applicable until you get to some really complicated code.

Entities in general

Entities are basically the objects a Schema is created to represent. As stated before, they have attributes and relationships. We will now go much further into explaining exactly what they are and how to write code for them.

There are two types of Entities: simpleType and complexType. A simpleType object has one value associated with it. A string is a perfect example of a simpleType object as it only contains the value of the string. Most simpleTypes used will be defined in the default Namespace; however, you can define your own simpleType at the bottom of the Schema (this will be brought up in the restrictions section). Because of this, the only objects you will most often need to include in your Schema are complexTypes. A complexType is an object with more than one attribute associated with it, and it may or may not have a child elements attached to it. Here is an example of a complexType object:

```xml
<xsd:complexType name="GenreType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="description" type="xsd:string"/>
    <xsd:element name="movie" type="MovieType" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```
This code begins with the declaration of a complexType and its name. When other entities refer to it, such as a parent element, it will refer to this name. The 2nd line begins the sequence of attributes and child elements, which are all declared as an "element". The elements are declared as elements with the 1st part of the line of code, and their name to which other documents will refer is included as the "name" as the 2nd part. After the first two declarations comes the "type" declaration. Note that for the name and description elements their type is "xsd:string" showing that the type string is defined in the Namespace "xsd". For the movie element, the type is "MovieType", and because there is no Namespace before "MovieType", it is assumed that this type is included in this Schema. (it could refer to a type defined in another Schema if the other Schema was included at the top of the Schema. don't worry about that now) "minOccurs" and "maxOccurs" represents the relationship between Genre's and MovieTypes. "minOccurs" can be either 0 or an arbitrary number, depending only on the data model. "maxOccurs" can be either 1 (a one to one relationship), an arbitrary number (a one to many relationship), or "unbounded" (a one to many relationship).

For each schema, there must be one root element. This entity contains every other entity underneath it in the hierarchy. For instance, when creating a schema to include a list of movies, the root element would be something like MovieDatabase, or maybe MovieCollection, just something that would logically contain all the other objects (like genre, movie, actor, director, plotline, etc.). It is always started with this line of code: `<xsd:element name="xxx">` showing that it is the root element and then goes on as a normal complexType. All other objects will begin with either simpleType or complexType. Here is sample code for a MovieDatabase root element:

```
<xs:complexType name="MovieDatabase">
  <xsd:sequence>
    <xsd:element name="Genre" type="GenreType" minOccurs="2" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Exhibit 4: The Root Element

This represents a MovieDatabase where the child element of MovieDatabase is a Genre. From there it goes onto movie, etc. We will continue to use this example help you better understand.

### The Parent / Child Relationship

The Parent / Child Relationship is a key topic in Data Schemas. It represents the basic structure of the data model's hierarchy by clearly laying out the top down configuration. Look at this piece of code which shows how movies have actors associated with them:

```
<xs:complexType name="MovieType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="actor" type="ActorType" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Exhibit 3: The complexType Element

Within each MovieType, there is an element named "actor" which is of "ActorType". When the XML document is populated with information, the surrounding tags for actor will be `<actor>` (actor) and not `<ActorType>`. To keep your Schema flowing smoothly and without error, the type field in the Parent Element will always equal the name field in the declaration of the complexType Child Element.

### Attributes and Restrictions

An attribute of an entity is a simpleType object in that it only contains one value. `<xsd:element name="lnname" type="xsd:string"/>` is a good example of an attribute. It is declared as an element, has a name associated with it, and has a type declaration. Located in the appendix of this chapter is a long list of simpleTypes built into the default Namespace. Attributes are incredibly simple to use, until you try and restrict them.

In some cases, certain data must abide by a standard to maintain data integrity. An example of this would be a Social Security number or an email address. If you have a database of email addresses that sends mass emails to, you would need all of them to be valid addresses, or else you’d get tons of error messages each time you send that mass email. To avoid this problem, you can essentially take a known simpleType and add a restriction to it to better suit your needs. Now you can do this two ways, but one is simpler and better to use in Data Schemas. You could edit the simpleType within its declaration in the Parent Element, but it gets messy, and if another Schema wants to use it, the code must be written again. The better way to do it is to list a new type at the bottom of the Schema that edits a previously known simpleType. Here is an example of this with a Social Security number:

```
<xs:simpleType name="ssnType">
  <xs:restriction base="xsd:string">
    <xs:pattern value="\d{3}‐\d{2}‐\d{4}"/>
  </xs:restriction>
</xs:simpleType>
```

Exhibit 6: Restriction on a simpleType

This was included in the Schema below the last Child Element and before the closing `<xsd:schema>`. The first line declares the simpleType and gives it a name, "ssnType". You could name yours anything you want, as long as you reference it correctly throughout the Schema. By doing this, you can use this type anywhere in the Schema, or anywhere in another Schema, provided the references are correct. The second line lets the Schema know it is a restricted type and its base is a string defined in the default Namespace. Basically, this is a string with a restriction on it, and the third line is the actual restriction. It can be one of many types of restrictions, which are listed in the Appendix of this chapter. This one happens to be of type "pattern". A "pattern" means that only a certain sequence of characters will be allowed in the XML document and is defined in the value field. This particular one means three digits, a hyphen, two digits, a hyphen, and four digits. To learn more about how to use restrictions, follow this link (http://www.w3schools.com/schema/schema_elements_ref.asp) to the W3 school's section on restrictions.
Not of little import: Introducing the `<xsd:import>` tag

The `<xsd:import>` tag is used to import a schema document and the namespace associated with the data types defined within the schema document. This allows an XML schema document to reference a type library using namespace names (prefixes). Let's take a closer look at a simple XML instance document for a store that uses these multiple namespace names:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<store:SimpleStore xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:MGR="http://www.opentourism.org/xmltext/CoreSchema"
    xsi:schemaLocation="http://www.opentourism.org/xmltext/Store.xsd">
    <!– Note the explicitly defined namespace declarations, the prefix store represents data types defined in the code:http://www.opentourism.org/xmltext/Store.xml namespace and the prefix MGR represents data types defined in the code:http://www.opentourism.org/xmltext/CoreSchema/code> namespace. Also, notice that there is no default namespace declaration - every element and attribute must be associated with a namespace (we will see this is necessary when we examine the schema document) -->
    <store:Store>
        <MGR:Name xmlns:MGR="http://www.opentourism.org/xmltext/CoreSchema">
            <MGR:FirstName>Michael</MGR:FirstName>
            <MGR:MiddleNames>Jay</MGR:MiddleNames>
            <MGR:LastName>Fox</MGR:LastName>
        </MGR:Name>
        <store:StoreName>The Gap</store:StoreName>
        <store:StoreAddress>
            <store:Street>86 Nowhere Ave.</store:Street>
            <store:City>Los Angeles</store:City>
            <store:State>CA</store:State>
        </store:StoreAddress>
    </store:Store>
</store:SimpleStore>
```

Exhibit 7 XML Instance Document – [1](http://www.opentourism.org/xmltext/store.xml)

Let's look at the schema document and see how the `<xsd:import>` tag was used to import data types from a type library (external schema document).

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.opentourism.org/xmltext/Store.xml"
    xmlns:MGR="http://www.opentourism.org/xmltext/CoreSchema"
    targetNamespace="http://www.opentourism.org/xmltext/Store.xml"
    elementFormDefault="qualified">
    <!– The prefix MGR is bound to the following namespace name: code:http://www.opentourism.org/xmltext/CoreSchema coco> namespace name, which was bound to the MGR prefix. The elementFormDefault attribute has the value ‘qualified’ indicating that an XML instance document must use qualified names for every element (default namespace can not be used) ––>
    <xsd:import namespace="http://www.opentourism.org/xmltext/CoreSchema"
        schemaLocation="ManagerTypeLib.xsd"/>
    <xsd:element name="SimpleStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Store" type="StoreType" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StoreType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="MGR:Name"/>;
                <xsd:element name="StoreName" type="xsd:string"/>
                <xsd:element name="StoreAddress" type="StoreAddressType"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StoreAddressType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element type="xsd:string"/>
                <xsd:element name="City" type="xsd:string"/>
                <xsd:element name="State" type="xsd:string"/>
                <xsd:element name="ZipCode" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Exhibit 8: XML Schema [http://www.opentourism.org/xmltext/SimpleStore.xsd]

Like the include tag and the redefine tag, the import tag is another means of incorporating any data types from an external schema document into another schema document and must occur before any element or attribute declarations. These mechanisms are important when XML schemas are modularized and type libraries are being maintained and used in multiple schema documents.

When the whole is greater than the sum of its parts:

Schema Modularization
Now that we have covered all three methods of incorporating external XML schemas, let’s consider the importance of these mechanisms. As is typical with most programming code, redundancy is frowned upon; this is true for custom data type definitions as well. If a custom data type already exists that can be applied to an element in your schema document, does it not make sense to use this data type rather than create it again within your new schema document? Moreover, if you know that a single data type can be reused for several applications, should you not have a method for referencing that data type when you need it?

The idea behind modular schemas is to examine what your schema does, determine what data types are frequently used in one form or another and develop a type library. As your needs for more complex schemas increase you can continue to add to your library, reuse data types in your type library, and redefine those data types as needed. An example of this reuse would be a schema for customer information – different departments would use different schemas as they would need only partial customer information. However most, if not all, departments would need some specific customer information, like name and contact information, which could be incorporated in the individual departmental schema documents.

Schema modularization is a “best practice”. By maintaining a type library and reusing and redefining types in the type library, you can help ensure that your XML schema documents don't become overwhelming and difficult to read. Readability is important, because you may not be the only one using these schemas, and it is important that others can easily understand your schema documents.

“Choose, but choose wisely…”: Schema alternatives

Thus far in this book we have only discussed XML schemas as defined by the World Wide Web Consortium (W3C). Yet there are other methods of defining the data contained within an XML instance document, but we will only mention the two most popular and well known alternatives: Document Type Definition (DTD) and Relax NG Schema.

We will cover DTDs in the next chapter. Relax NG schema is a newer and has many of the same features that W3C XML schema have; Relax NG also claims to be simpler, and easier to learn, but this is very subjective. For more about Relax NG, visit: http://www.relaxng.org/

Appendix

First is the full Schema used in the examples throughout this chapter:

```xml
<?xml version="1.0" encoding="UTF‐8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified">

    <xsd:element name="MovieDatabase">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Genre" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
                <xsd:element name="Movie" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
                <xsd:element name="Actor" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="GenreType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="description" type="xsd:string"/>
                <xsd:element name="movies" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="MovieType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="rating" type="xsd:string"/>
                <xsd:element name="director" type="xsd:string"/>
                <xsd:element name="writer" type="xsd:string"/>
                <xsd:element name="year" type="xsd:int"/>
                <xsd:element name="tagline" type="xsd:string"/>
                <xsd:element name="actor" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="ActorType">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="fname" type="xsd:string"/>
                <xsd:element name="lname" type="xsd:string"/>
                <xsd:element name="gender" type="xsd:string"/>
                <xsd:element name="bday" type="xsd:string"/>
                <xsd:element name="birthplace" type="xsd:string"/>
                <xsd:element name="tagline" type="xsd:string"/>
                <xsd:element name="known_for" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:simpleType name="xssType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="d[3,4]-d[3,6]-d[4,8]/"></xsd:restriction>
        </xsd:simpleType>
    </xsd:simpleType>

</xsd:schema>
```

It’s time to go back to the beginning… and review all of the schema data types, elements, and attributes that we have covered thus far (and maybe a few that we have not). The following tables will detail the XML data types, elements and attributes that can be used in an XML Schema.

### Primitive Types

This is a table with all the primitive types the attributes in your schema can be.

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>Legal example</th>
<th>Constraining facets</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:anyURI</td>
<td><code>&lt;xsd:element name = “url” type = “xsd:anyURI” /&gt;</code></td>
<td><a href="http://www.w3.com">http://www.w3.com</a></td>
<td>length, minLength, maxLength, pattern, enumeration, whitespace</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td><code>&lt;xsd:element name = “hasChildren” type = “xsd:boolean” /&gt;</code></td>
<td>true or false or 1 or 0</td>
<td>pattern and whitespace</td>
</tr>
<tr>
<td>xsd:byte</td>
<td><code>&lt;xsd:element name = “stdDev” type = “xsd:byte” /&gt;</code></td>
<td>-128 through 127</td>
<td>length, minInclusive, maxInclusive, minExclusive, maxExclusive,</td>
</tr>
</tbody>
</table>
Here is a list of all the elements which can be included in your schemas.

<table>
<thead>
<tr>
<th>Element</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Specifies that the child elements can appear in any order. Each child element can occur 0 or 1 time</td>
</tr>
<tr>
<td>annotation</td>
<td>Specifies the top-level element for schema comments</td>
</tr>
<tr>
<td>any</td>
<td>Enables the author to extend the XML document with elements not specified by the schema</td>
</tr>
<tr>
<td>any:Attribute</td>
<td>Enables the author to extend the XML document with attributes not specified by the schema</td>
</tr>
<tr>
<td>appInfo</td>
<td>Specifies information to be used by the application (must go inside annotation)</td>
</tr>
</tbody>
</table>

Here is a list of all the elements which can be included in your schemas.

<table>
<thead>
<tr>
<th>Element</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Specifies that the child elements can appear in any order. Each child element can occur 0 or 1 time</td>
</tr>
<tr>
<td>annotation</td>
<td>Specifies the top-level element for schema comments</td>
</tr>
<tr>
<td>any</td>
<td>Enables the author to extend the XML document with elements not specified by the schema</td>
</tr>
<tr>
<td>any:Attribute</td>
<td>Enables the author to extend the XML document with attributes not specified by the schema</td>
</tr>
<tr>
<td>appInfo</td>
<td>Specifies information to be used by the application (must go inside annotation)</td>
</tr>
<tr>
<td>attribute</td>
<td>Defines an attribute</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>Defines an attribute group to be used in complex type definitions</td>
</tr>
<tr>
<td>choice</td>
<td>Allows only one of the elements contained in the &lt;choice&gt; declaration to be present within the containing element</td>
</tr>
<tr>
<td>complexContent</td>
<td>Defines extensions or restrictions on a complex type that contains mixed content or elements only</td>
</tr>
<tr>
<td>complexType</td>
<td>Defines a complex type element</td>
</tr>
<tr>
<td>documentation</td>
<td>Defines text comments in a schema (must go inside annotation)</td>
</tr>
<tr>
<td>element</td>
<td>Defines an element</td>
</tr>
<tr>
<td>extension</td>
<td>Extends an existing simpleType or complexType element</td>
</tr>
<tr>
<td>field</td>
<td>Specifies an XPath expression that specifies the value used to define an identity constraint</td>
</tr>
<tr>
<td>group</td>
<td>Defines a group of elements to be used in complex type definitions</td>
</tr>
<tr>
<td>import</td>
<td>Adds multiple schemas with different target namespace to a document</td>
</tr>
<tr>
<td>include</td>
<td>Adds multiple schemas with the same target namespace to a document</td>
</tr>
<tr>
<td>key</td>
<td>Specifies an attribute or element value as a key (unique, non-nullable, and always present) within the containing element in an instance document</td>
</tr>
<tr>
<td>keyref</td>
<td>Specifies that an attribute or element value correspond to those of the specified key or unique element</td>
</tr>
<tr>
<td>list</td>
<td>Defines a simple type element as a list of values</td>
</tr>
<tr>
<td>notation</td>
<td>Describes the format of non-XML data within an XML document</td>
</tr>
<tr>
<td>redefine</td>
<td>Redefines simple and complex types, groups, and attribute groups from an external schema</td>
</tr>
<tr>
<td>restriction</td>
<td>Defines restrictions on a simpleType, simpleContent, or a complexContent</td>
</tr>
<tr>
<td>schema</td>
<td>Defines the root element of a schema</td>
</tr>
<tr>
<td>selector</td>
<td>Specifies an XPath expression that selects a set of elements for an identity constraint</td>
</tr>
<tr>
<td>sequence</td>
<td>Specifies that the child elements must appear in a sequence. Each child element can occur from 0 to any number of times</td>
</tr>
<tr>
<td>simpleContent</td>
<td>Contains extensions or restrictions on a text-only complex type or on a simple type as content and contains no elements</td>
</tr>
<tr>
<td>simpleType</td>
<td>Defines a simple type and specifies the constraints and information about the values of attributes or text-only elements</td>
</tr>
<tr>
<td>union</td>
<td>Defines a simple type as a collection (union) of values from specified simple data types</td>
</tr>
<tr>
<td>unique</td>
<td>Defines that an element or an attribute value must be unique within the scope</td>
</tr>
</tbody>
</table>

**Schema Restrictions and Facets for data types**

( from http://www.w3schools.com/schema/schema_elements_ref.asp )

Here is a list of all the types of restrictions which can be included in your schema.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumeration</td>
<td>Defines a list of acceptable values</td>
</tr>
<tr>
<td>fractionDigits</td>
<td>Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero</td>
</tr>
<tr>
<td>length</td>
<td>Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>Specifies the upper bounds for numeric values (the value must be less than this value)</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>Specifies the upper bounds for numeric values (the value must be less than or equal to this value)</td>
</tr>
<tr>
<td>maxLength</td>
<td>Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero</td>
</tr>
<tr>
<td>minExclusive</td>
<td>Specifies the lower bounds for numeric values (the value must be greater than this value)</td>
</tr>
<tr>
<td>minInclusive</td>
<td>Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)</td>
</tr>
<tr>
<td>minLength</td>
<td>Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero</td>
</tr>
<tr>
<td>pattern</td>
<td>Defines the exact sequence of characters that are acceptable</td>
</tr>
<tr>
<td>totalDigits</td>
<td>Specifies the exact number of digits allowed. Must be greater than zero</td>
</tr>
<tr>
<td>whiteSpace</td>
<td>Specifies how white space (line feeds, tabs, spaces, and carriage returns) are handled</td>
</tr>
</tbody>
</table>

**Regex**

Special regular expression (regex) language can be used to construct a pattern. The regex language in XML Schema is based on Perl's regular expression language. The following are some common notations:

- `.` (the period) for any character at all
- `d` for any digit
- `D` for any non-digit
- `w` for any word (alphanumeric) character
- `W` for any non-word character (i.e. -, *, =)
for any white space (including space, tab, newline, and return)
S
for any character that is not white space
x*
to have zero or more x’s
(xy)*
to have zero or more xy’s
x+
repetition of the x, at least once
x?
to have one or zero x’s
(xy)?
to have one or no xy’s
[abc]
to include one of a group of values
[0-9]
to include the range of values from 0 to 9
x[5]
to have exactly 5 x’s (in a row)
x[5,]
to have at least 5 x’s (in a row)
x[5,8]
at least 5 but at most 8 x’s (in a row)
(xy)[2]
to have exactly 2 xy’s (in a row)

For example, the pattern for validating a Social Security Number is \d{3}-\d{2}-\d{4}
The schema code for emailAddressType is \w+\W*\w*@\{1\}\w+\W*\w*\.

[w+]
at least one word (alphanumeric) character, e.g. answer
[W+]
followed by one, or many non-word character(s), e.g. -
w*[@{1}]
followed by any (or none) word character and at-sign, e.g. my@)
w+
followed by at least one word character, e.g. mail
[W+]
followed by one, or many non-word character(s), e.g. 
w+
followed by at least one word character and period, e.g. please,
w+.*
zero to infinite times followed by the previous string, e.g. opentourism.
w*
finally followed by none, one or many word character(s) e.g. org

email-address: answer-my@mail please.opentourism.org

Instance Document Attributes
These attributes do NOT need to be declared within the schemas

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsi:nil</td>
<td>Indicates that a certain element does not have a value or that the value is unknown. The element must be set to nillable inside the schema document:</td>
<td>&lt;full_name xmlns:xsi= &quot;<a href="http://www.w3.org/2001/XMLSchema-instance%22%3E">http://www.w3.org/2001/XMLSchema-instance&quot;&gt;</a> &lt;first_name&gt;Madonna&lt;/first_name&gt; &lt;last_name xsi:nil=&quot;true&quot;/&gt; &lt;/full_name&gt;</td>
</tr>
<tr>
<td>xsi:noNamespaceSchemaLocation</td>
<td>Locates the schema for elements that are not in any namespace</td>
<td>&lt;radio xmlns:xsi:noNamespaceSchemaLocation= &quot;http://www.opentourism.org/xmtext/radio.xsd&quot;&gt; &lt;!—radio stuff goes here &gt; &lt;/radio&gt;</td>
</tr>
<tr>
<td>xsi:type</td>
<td>Can be used in instance documents to indicate the type of an element.</td>
<td>&lt;height xsi:type:xsd:decimal=&quot;&quot;&gt;78.9&lt;/height&gt;</td>
</tr>
</tbody>
</table>

For more information on XML Schema structures, data types, and tools you can visit http://www.w3.org/XML/Schema.
In previous chapters, we have learned how to generate HTML documents from XML documents and XSL stylesheets. In this chapter, we will learn how to convert those HTML documents into valid XHTML. We will discuss why XHTML has evolved as a standard and when it should be used.

The Evolution of XHTML

Originally, Web pages were designed in HTML. Unfortunately most implementations of this markup language allow all sorts of mistakes and bad formatting. Major browsers were designed to be forgiving, and poor code would display with few problems in most cases. This poor code was often not portable between browsers, e.g. a page would render in Netscape but not Internet Explorer or vice versa. The accounting for human error and bad formatting takes an amount of processing power that small handheld devices might not have. Thus when displaying data on handhelds, a tiny mistake can crash the device.

XHTML partially mitigates these problems. The processing burden is reduced by requiring XHTML documents to conform to the much stricter rules defined in XML. Aside from the stricter rules, HTML 4.01 and XHTML 1.0 are functionally equivalent. If a document breaks XML's well-formedness rules, an XHTML-compliant browser must not render the page. If a document is well-formed but invalid, an XHTML-compliant browser may render the page, so a significant number of mistakes still slip through.

In this chapter, we will examine in detail how to create an XHTML document.

The biggest problem with HTML from a design standpoint is that it was never meant to be a graphical design language. The original version of HTML was intended to structure human readable content (e.g. marking a section of text as a paragraph), not to format it (e.g. this paragraph should be displayed in 14pt Arial). HTML has evolved far past its original purpose and is being stretched and manipulated to cover cases that the original HTML designers never imagined.

The recommended solution is to use a separate language to describe the presentation of a group of documents. Cascading Style Sheets (CSS) is a language used for describing presentation. From version 1.1 of XHTML upwards web pages must be formatted using CSS or a language with equivalent capabilities such as XSLT (XSL Transformations). The use of CSS or XSLT is optional in XHTML 1.0 unless the strict variant is used. HTML 4.01 supports CSS but not XSLT.

So What is XHTML?

As you might have guessed, XHTML stands for eXtensible HyperText Markup Language. It is a cross between HTML and XML. It fulfills two major purposes that were ignored by HTML:

1. XHTML is a stricter standard than HTML. XHTML documents must be well-formed just like regular XML. This reduces vagaries and inconsistency between browsers, because browsers do not have to decide how to display a badly-formed page. Malformed XHTML is not allowed.

   Note 1: Browsers only enforce well-formedness if the MIME type is set to application/xhtml+xml. If the MIME type is set to text/html, the browser will allow badly-formed documents. There are a large number of 'XHTML' documents on the web that are badly-formed and get away with it because their MIME type is text/html.

   Note 2: Browsers are not required to check for validity. See Invalid XHTML below for an example.

2. XHTML allows for modularization (m12n). For different environments different element and attribute subsets can be defined.

The best thing about XHTML is that it is almost the same as HTML! If you know how to write an HTML document, it will be very simple for you to create an XHTML document without too much trouble. The biggest thing that you must keep in mind is that unlike with HTML, where simple errors like missing a closing tag are ignored by the browser, XHTML code must be written according to an exact specification. We will see later that adhering to these strict specifications actually allows XHTML to be more flexible than HTML.

XHTML Document Structure

At a minimum, an XHTML document must contain a DOCTYPE declaration and four elements: html, head, title, and body:

```xml
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="...">
  <head>
    <title></title>
  </head>
  <body></body>
</html>
```

The opening html tag of an XHTML document must include a namespace declaration for the XHTML namespace. The DOCTYPE declaration should appear immediately before the html tag in an XHTML document. It can follow one of three formats.

XHTML 1.0 Strict

```xml
<!DOCTYPE html PUBLIC ".//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The Strict declaration is the least forgiving. This is the preferred DOCTYPE for new documents. Strict documents tend to be streamlined and clean. All formatting will appear in Cascading Style Sheets rather than the document itself. Elements that should be included in the Cascading Style Sheet and not the document itself include, but are not limited to:

```html
<body text="blue", counderline/italic/... >
```

There are also certain instances where your code needs to be nested within block elements.

Incorrect Example:

```html
<p>I hope that you enjoy///your stay.</p>
```

Correct Example:

```html
<p> I hope that you enjoy/// your stay.</p>
```
Hoping that you enjoy your stay.

XHTML 1.0 Transitional

This declaration is intended as a halfway house for migrating legacy HTML documents to XHTML 1.0 Strict. The W3C encourages authors to use the Strict DOCTYPE for new documents. (The XHTML 1.0 Transitional DTD refers readers to the relevant note in the HTML4.01 Transitional DTD.)

This DOCTYPE does not require CSS for formatting; although, it is recommended. It generally tolerates inline elements found where block-level elements are expected.

There are a couple of reasons why you might choose this DOCTYPE for new documents.

- You require backwards compatibility with browsers that support the formatting elements of XHTML but do not support CSS. This is a very small fraction of general users (less than 1%). Many browsers that don't support CSS don't support HTML 4.0 or XHTML either. However, it may be useful on a corporate intranet that has a larger than normal fraction of very old (pre-2000) browsers.
- You need to link to frames. Using frames is discouraged as they work badly in many browsers.

XHTML 1.0 Frameset

If you are creating a page with frames, this declaration is appropriate. However, since frames are generally discouraged when designing Web pages, this declaration should be used rarely.

XML Prolog

Additionally, XHTML authors are encouraged by the W3C to include the following processing instruction as the first line of each document:

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

Although it is recommended by the standard, this processing instruction may cause errors in older Web browsers including Internet Explorer version 6. It is up to the individual author to decide whether to include the prolog.

Language

It is good practice to include the optional xml:lang attribute [2] on the html element to describe the document's primary language. For compatibility with HTML the lang attribute should also be specified with the same value. For an English language document use:

```html
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

The xml:lang and lang attributes can also be specified on other elements to indicate changes of language within the document, e.g. a French quotation in an English document.

Converting HTML to XHTML

In this section, we will discover how to transform an HTML document into an XHTML document. We will examine each of the following rules:

- Documents must be well-formed
  - Tags must be properly nested
  - Elements must be closed
- Tags must be lowercase
- Attribute names must be lowercase
- Attribute values must be quoted
- Attributes cannot be minimized
- The name attribute is replaced with the id attribute (in XHTML 1.0 both name and id should be used with the same value to maintain backwards-compatibility).
- Plain ampersands are not allowed
- Scripts and CSS must be escaped (enclose them within the tags `<![CDATA[ and ]]>`) or preferably moved into external files.

Documents must be well-formed

Because XHTML conforms to all XML standards, an XHTML document must be well-formed according to the W3C's recommendations for an XML document. Several of the rules here reemphasize this point. We will consider both incorrect and correct examples.

Tags must be properly nested

Browsers widely tolerate badly nested tags in HTML documents.

```html
<b><u>This text is probably bold and underlined, but inside incorrectly nested tags.
</u></b>
```
The text above would display as bold and underlined, even though the end tags are not in the proper order. An XHTML page will not display if the tags are improperly nested, because it would not be considered a valid XML document. The problem can be easily fixed.

```
<b><u>This text is bold and underlined and inside properly nested tags.</u></b>
```

**Elements must be closed**

Again, XHTML documents must be considered valid XML documents. For this reason, all tags must be closed. HTML specifications listed some tags as having "optional" end tags, such as the `<p>` and `<li>` tags.

```
<p>Here is a list:
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
</p>
```

In XHTML, the end tags must be included.

```
<p>Here is a list: </p>
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

What should we do about HTML tags that do not have a closing tag? Some special tags do not require or imply a closing tag.

```
<img src="titlebar.gif" alt="Title">
<br>
<br>
<p> Welcome to my web page!</p>
```

In XHTML, the XML rule of including a closing slash within the tag must be followed.

```
<img src="titlebar.gif" alt="title" />
<br /> 
<br />
<p>Welcome to my web page!</p>
```

Note that some of today's browsers will incorrectly render a page if the closing slash does not have a space before it (`<br/>`). Although it is not part of the official recommendation, you should always include the space (`<br />`) for compatibility purposes.

Here are the common empty tags in HTML:

- `area`
- `base`
- `basefont`
- `br`
- `hr`
- `img`
- `input`
- `link`
- `meta`
- `param`

**Tags must be lowercase**

In HTML, tags could be written in either lowercase or uppercase. In fact, some Web authors preferred to write tags in uppercase to make them easier to read. XHTML requires that all tags be lowercase.

```
<H1>This is an example of bad case.</H1>
```

This difference is necessary because XML differentiates between cases. XML would read `<H1>` and `<h1>` as different tags, causing problems in the above example.

```
<h1>This is an example of good case.</h1>
```

The problem can be easily fixed by changing all tags to lowercase.

**Attribute names must be lowercase**

Following the pattern of writing all tags in lowercase, all attribute names must also be in lowercase.

```
<p_class="specialText">Important Notice</p>
```

The correct tags are easy to create.

```
<p class="specialText">Important Notice</p>
```

**Attribute values must be quoted**
Some HTML values do not require quotation marks around them. They are understood by browsers.

```html
<table border=1 width=100%>
</table>
```

XHTML requires all attributes to be quoted. Even numeric, percentage, and hexadecimal values must appear in quotations for them to be considered part of a proper XHTML document.

```html
<table border="1" width="100%">
</table>
```

**Attributes cannot be minimized**

HTML allowed some attributes to be written in shorthand, such as selected or noresize.

```html
<form>
<input checked ...
<input disabled ...
</form>
```

When using XHTML, attribute minimization is forbidden. Instead, use the syntax `x="x"`, where `x` is the attribute that was formerly minimized.

```html
<form>
<input checked="checked" ...
<input disabled="disabled" ...
</form>
```

A complete list of minimized attributes follows:

- checked
- compact
- declare
- defer
- disabled
- ismap
- nohref
- noscroll
- noresize
- nowrap
- readonly
- selected
- multiple

**The name attribute is replaced with the id attribute**

HTML 4.01 standards define a name attribute for the tags `a`, `applet`, `frame`, `iframe`, `img`, and `map`.

```html
<a name="anchor">
<img src="banner.gif" name="mybanner" />
</a>
```

XHTML has deprecated the name attribute. Instead, the id attribute is used. However, to ensure backwards compatibility with today's browsers, it is best to use both the name and id attributes.

```html
<a name="anchor" id="anchor">
<img src="banner.gif" name="mybanner" id="mybanner" />
</a>
```

As technology advances, it will eventually be unnecessary to use both attributes and XHTML 1.1 removed name altogether.

**Ampersands are not supported**

Ampersands are illegal in XHTML.

```html
<a href="home.aspx?status=done&amp;ItWorked=false">Home &amp; Garden</a>
```

They must instead be replaced with the equivalent character code `&`.  

```html
<a href="home.aspx?status=done&amp;ItWorked=false">Home &amp; Garden</a>
```

**Image alt attributes are mandatory**

Because XHTML is designed to be viewed on different types of devices, some of which are not image-capable, alt attributes must be included for all images.

```html
<img src="titlebar.gif" alt="title" />
```

Remember that the img tag must include a closing slash in XHTML!
Scripts and CSS must be escaped

Internal scripts and CSS often include characters like the ampersand and less-than characters.

```html
<script language="JavaScript">
    <!-‐‐
    document.write('Hello World!');
    //‐‐>
</script>
```

If you are using internal scripts or CSS, enclose them within the tags `<![CDATA[ and ]]>`. This will mark them as character data that should not be parsed. If you do not use these tags, characters like & and < will be treated as start-of-character entities (like &nbsp;) and tags (like <b>) respectively. This will cause your page to behave unpredictably, and it may invalidate your code.

Additionally, the type attribute is mandatory for scripts. The comment tags <!-- and --> that have traditionally been used to hide JavaScript from noncompliant browsers should not be included. The XML standard states that text enclosed in comment tags may be completely excluded from rendered documents, which would lose all script enclosed in the tags.

```javascript
<script type="text/javascript" language="javascript">
    /*!
    <![CDATA[
    document.write('Hello World!');
    /*]]>*/
</script>
```

Also `document.write();` is not permitted in XHTML documents. You must used node creation methods such as `document.createElementNS();` instead. Confusingly, `document.write();` will appear to work as expected if the document is incorrectly served with a MIME type of `text/html` (the type for HTML documents), instead of `application/xhtml+xml` (the type for XHTML documents). If the MIME type is `text/html` the document will be parsed as HTML which allows `document.write();`. Parsing the document as HTML defeats the purpose of writing it in XHTML.

Similar changes must be made for internal styleheets.

```css
<style type="text/css">
    <!-‐‐
    .SpecialClass {  
    color: #000000;
    }  
    //‐‐>
</style>
```

The type attribute must be included, and the CDATA tags should be used.

```css
<style type="text/css">
    <![CDATA[
    .SpecialClass {  
    color: #000000;
    }  
    ]]>*/
</style>
```

Because scripts and CSS may complicate an XHTML document, it is strongly recommended that they be placed in external .js and .css files, respectively. They can then be linked to from your XHTML document.

```html
<script src="myscript.js" type="text/javascript" />
<link href="styles.css" type="text/css" rel="stylesheet" />
```

Some elements may not be nested

The W3C recommendations state that certain elements may not be contained within others in an XHTML document, even when no XML rules are violated by the inclusion. Elements affected are listed below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Cannot contain ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>pre</td>
<td>big, img, object, small, sub, sup</td>
</tr>
<tr>
<td>button</td>
<td>button, fieldset, form, iframe, input, isindex, label, select, textarea</td>
</tr>
<tr>
<td>label</td>
<td>label</td>
</tr>
<tr>
<td>form</td>
<td>form</td>
</tr>
</tbody>
</table>

When to convert

By now, it probably sounds as though converting an HTML document into XHTML is easy, but tedious. When would you want to convert your existing pages into XHTML? Before deciding to change your entire Web site, consider these questions.

- **Do you want your pages to be easily viewed over a nontraditional Internet-capable device, such as a PDA or Web-enabled telephone?** Will this be a goal of your site in the future? XHTML is the language of choice for Web-enabled portable devices. Now may be a good time for you to commit to creating an all-XHTML site.
- **Do you plan to work with XML in the future?** If so, XHTML may be a logical place to begin. If you head up a team of designers who are accustomed to using HTML, XHTML is a small step away. It may be less intimidating for beginners to learn XHTML than it is to try teaching them all about XML from scratch.
- **Is it important that your site be current with the most recent W3C standards?** Staying on top of current standards will make your site more stable and help you stay updated in the future, as you will only have to make small changes to upgrade your site to the newest versions of XHTML as they are approved by the W3C.
- **Will you need to convert your documents to another format?** As a valid XML document, XHTML can utilize XSL to be converted into text, plain HTML, another XHTML document, or another XML document. HTML cannot be used for this purpose.
MIME Types

XHTML 1.0 documents should be served with a MIME Type of application/xhtml+xml to Web browsers that can accept this type. XHTML 1.0 may be served with the MIME type text/html to clients that cannot accept application/xhtml+xml provided that the XHTML complies with the additional constraints in [Appendix C (http://www.w3.org/TR/html/#guidelines)] of the XHTML 1.0 specification. If you cannot configure your Web server to serve documents as different MIME types, you probably should not convert your Web site to XHTML.

You should check that your XHTML documents are served correctly to browsers that support application/xhtml+xml, e.g. Mozilla Firefox. Use 'Page Info' to verify that the type is correct.

XHTML 1.1 documents are often not backwards compatible with HTML and should not be served with a MIME type of text/html.[3] (http://www.w3.org/TR/xhtml-media-types/#summary)

Help Converting

- Strict Alternatives to Deprecated Attributes for Elements in XHTML 1.0 Strict (http://aamcf.co.uk/html/attr)

HTML Tidy

When creating HTML, it's very easy to make a mistake by leaving out an end tag or not properly nesting tags. HTML Tidy is a wonderful application that can be used to correct a number of errors with poorly formed HTML documents and convert it into XHTML. Tidy can also format ugly code to be more readable, including code generated by WYSIWYG editors. HTML Tidy can't generate clean code when it encounters problems it isn't sure of how to fix. In these cases, it will generate an error to let you know where the mistake is located in your document.

A few examples of problems that HTML Tidy can remedy:

- Missing or mismatched end tags.
- Improperly nested elements.
- Mixed up tags.
- Add a missing "/>" to properly close tags.
- Insert missing tags into lists.
- Add missing quotes around attribute values.
- Ability to insert the correct DOCTYPE value based on your code (can also recognize and report proprietary elements).

HTML Tidy can also be customized at runtime using a wide array of command line arguments. It is capable of indenting code to make it more readable as well as replacing FONT, NOBR, and CENTER tags with style tags and rules using CSS. Tidy can also be taught new tags by declaring them in the configuration file.

You can read more about HTML Tidy at the W3C's HTML Tidy (http://www.w3.org/People/Raggett/tidy/) site, as well as download the application as a binary or get the source code. There are several sites that offer HTML Tidy as an online service including the W3C (http://cgi.w3.org/cgi-bin/tidy) and Site Valet (http://valet.htmlhelp.com/tidy/).

You can also validate your page using the validator available at http://validator.w3.org/.

When not to convert

You shouldn't convert your Web pages if they will always be served with a MIME type of text/html. Make sure you know how to configure your server or server-side script to perform HTTP content negotiation so that XHTML capable browsers receive XHTML marked as application/xhtml+xml. If you can't set up content negotiation, stick to HTML 4.01. People viewing your Web pages with mainstream browsers will be unable to tell the difference between a valid HTML 4.01 web page and a valid XHTML 1.0 Web page.

Make sure the automated tests you run on your site simulate connections from both XHTML-compatible browsers, e.g. Mozilla Firefox, and non-XHTML-compatible browsers, e.g. Internet Explorer 6.0. This is particularly important if you use Javascript on your Web site. If maintaining two copies of your test suite is too time consuming, don't convert.

Bear in mind that valid HTML 4.01 Strict documents generally require less effort to convert to XHTML 1.1 than valid XHTML 1.0 Transitional documents. A valid HTML 4.01 Strict document can only contain elements that are valid in XHTML 1.1, although a few attributes may need changing. XHTML 1.0 Transitional documents on the other hand can contain ten element types and more than a dozen attributes that are not valid in XHTML 1.1. The XHTML 1.0 Transitional body element alone has six attributes that are not supported in XHTML 1.1.

Don't be pressured into using XHTML by people talking vaguely about bad practice. Pin them down to what they mean by bad practice. If they start talking about separation of content and presentation, they have confused the differences between HTML and XHTML with the differences between the Transitional and Strict doctypes. Both XHTML 1.0 Transitional and HTML 4.01 Transitional allow you to mix presentation and content in the same document, i.e. they allow this type of bad practice. Both HTML 4.01 Strict and XHTML 1.0 Strict force you to move the bulk of the presentation (but not all of it) in to CSS or an equivalent language. All four doctypes allow you to use embedded stylesheets, whereas, true separation requires that all CSS and Javascript be moved to external files.

XHTML 1.1

XHTML 1.0 is a suitable markup language for most purposes. It provides the option to separate content and presentation, which fits the needs of most Web authors. XHTML 1.1 enforces the separation of content and presentation. All deprecated elements and attributes have been removed. It also removes two attributes that were retained in XHTML 1.0 purely for backwards compatibility. The lang attribute is replaced by xml:lang and eaf= is replaced by id. Finally it adds support for ruby text found in East Asian documents.

DOCTYPE

The DOCTYPE for XHTML 1.1 is:

```
<!DOCTYPE html PUBLIC "‐//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Modularization
The modularization of XHTML, or XHTML m12n, provides suggestions for customizing XHTML, either by integrating subsets of XHTML into other XML applications or extending the XHTML element set. The framework defines two processes:

- How to group elements and attributes into "modules"
- How to combine modules to create new markup languages

The resulting languages, which the W3C calls "XHTML Host Languages", are based on the familiar XHTML structure but specialized for specific purposes. XHTML 1.1 is an example of a host language. It was created by grouping the different elements available to XHTML.

XHTML variations, while possible in theory, have not been widely adopted. There is continuing work being done to develop host languages, but their details are beyond the scope of this discussion.

Invalid XHTML

XHTML-compliant browsers are allowed to render invalid XHTML documents provided that the documents are well-formed. A simple example is given below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "‐//W3C//DTD XHTML 1.0 Strict//EN" 
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Invalid XHTML</title>
</head>
<body>
p
this sentence contains a <p>nested paragraph.</p>
</body>
</html>
```

Save the example as invalid.xhtml (the .xhtml extension is important) and open the page with Mozilla Firefox. The page will render even though it is invalid.

Summary

XHTML stands for eXtensible HyperText Markup Language. XHTML is very similar to HTML, but it is stricter and easier to parse. XHTML documents must be well-formed just like regular XML. XHTML allows for modularization. XHTML code must be written according to an exact specification unlike with HTML, where simple errors like missing a closing tag are ignored by the browser. Adhering to these strict specifications actually allows XHTML to be more flexible than HTML. The benefits described in this summary are only gained if the MIME type of the document is application/xhtml+xml. XHTML documents can be validated but most browsers choose not to.

Exercises

Answers
Throughout the previous chapters you have learned the basic concepts of XSL and how you must refer to nodes in an XML document when performing an XSL transformation. Up to this point you have been using a straightforward syntax for referring to nodes in an XML document. Although the syntax you have used so far has been XPath there are many more functions and capabilities that you will learn in this chapter. As you begin to comprehend how path language is used for referring to nodes in an XML document your understanding of XML as a tree structure will begin to fall into place. This chapter contains examples that demonstrate many of the common uses of XPath, but for the full XPath specification, see the latest version of the standard at:

http://www.w3.org/TR/xpath

XSL uses XPath heavily.

XPath

When you go to copy a file or 'cd' into a directory at a command prompt you often type something along the lines of ‘/home/darnell’ to refer to folders. This enables you to change into or refer to folders throughout your computer’s file system. XML has a similar way of referring to elements in an XML document. This special syntax is called XPath, which is short for XML Path Language.

XPath is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document.

XPath, although used for referring to nodes in an XML tree, is not itself written in XML. This was a wise choice on the part of the W3C, because trying to specify path information in XML would be a very cumbersome task. Any characters that form XML syntax would need to be escaped so that it is not confused with XML when being processed. XPath is also very succinct, allowing you to call upon nodes in the XML tree with a great degree of specificity without being unnecessarily verbose.

XML as a tree structure

The great benefit about XML is that the document itself describes the structure of data. If any of you have researched your family history, you have probably come across a family tree. At the top of the tree is some early ancestor and at the bottom of the tree are the latest children.

With a tree structure you can see which children belong to which parents, which grandchildren belong to which grandparents and many other relationships.

The neat thing about XML is that it also fits nicely into this tree structure, often referred to as an XML Tree.

Understanding node relationships

We will use the following example to demonstrate the different node relationships.

```
<bookstore>
  <book>
    <title>Less Than Zero</title>
    <author>Bret Easton Ellis</author>
    <year>1985</year>
    <price>13.95</price>
  </book>
</bookstore>
```

Parent

Each element and attribute has one parent.

The book element is the parent of the title, author, year, and price:

```
<book>
  <title>Less Than Zero</title>
  <author>Bret Easton Ellis</author>
  <year>1985</year>
  <price>13.95</price>
</book>
```

Children

Element nodes may have zero, one or more children.

The title, author, year, and price elements are all children of the book element:

```
<book>
  <title>Less Than Zero</title>
  <author>Bret Easton Ellis</author>
  <year>1985</year>
  <price>13.95</price>
</book>
```

Siblings

Nodes that have the same parent.

The title, author, year, and price elements are all siblings:

```
<book>
  <title>Less Than Zero</title>
  <author>Bret Easton Ellis</author>
  <year>1985</year>
  <price>13.95</price>
</book>
```

Ancestors

A node's parent, parent's parent, etc.

The ancestors of the title element are the book element and the bookstore element:

```
<book>
  <title>Less Than Zero</title>
  <author>Bret Easton Ellis</author>
  <year>1985</year>
  <price>13.95</price>
</book>
```

Descendants

A node's children, children's children, etc.

Descendants of the bookstore element are the book, title, author, year, and price elements:

```
<bookstore>
  <book>
    <title>Less Than Zero</title>
    <author>Bret Easton Ellis</author>
    <year>1985</year>
    <price>13.95</price>
  </book>
</bookstore>
```

Also, it is still useful in some ways to think of an XML file as simultaneously being a serialized file, like you would view it in an XML editor. This is so you can understand the concepts of preceding and following nodes. A node is said to precede another if the original node is before the other in document order. Likewise, a node follows another if it is after that node in document order. Ancestors and descendants are not considered to be either preceding or following a node. This concept will come in handy later when discussing the concept of an axis.

Abbreviated vs. Unabbreviated XPath syntax

XPath was created so that nodes can be referred to very succinctly, while retaining the ability to search on many options. Most uses of XPath will involve searching for child nodes, parent nodes, or attribute nodes of a particular node. Because these uses are so common, an abbreviated syntax can be used to refer to these commonly-searched nodes. Following is an XML document that simulates a tree (the type that has leaves and branches.) It will be used to demonstrate the different types of syntax.

```
<?xml version="1.0" encoding="UTF-8"?>
<trunk name="the_trunk">
  <bigBranch name="bb1" thickness="thick">
    <smallBranch name="sb1">
      <leaf name="leaf1" color="brown"/>
      <leaf name="leaf2" weight="50"/>
    </smallBranch>
    <smallBranch name="sb2">
      <leaf name="leaf4" weight="90"/>
      <leaf name="leaf5" color="purple"/>
    </smallBranch>
  </bigBranch>
</trunk>
```
Following are a few examples of XPath location paths in English, Abbreviated XPath, then Unabbreviated XPath.

Selection 1:

**English:** All `<leaf>` elements in this document that are children of `<smallBranch>` elements that are children of `<bigBranch>` elements, that are children of the trunk, which is a child of the root.

**Abbreviated:** `trunk/bigBranch/smallBranch/leaf`

**Unabbreviated:** `/child::trunk/child::bigBranch/child::smallBranch/child::leaf`

Selection 2:

**English:** The `<bigBranch>` elements with ‘name’ attribute equal to ‘bb3,’ that are children of the trunk element, which is a child of the root.

**Abbreviated:** `trunk/bigBranch[@name='bb3']`

**Unabbreviated:** `/child::trunk/child::bigBranch[attribute::name='bb3']`

Notice how we can specify which bigBranch objects we want by using a predicate in the previous example. This narrows the search down to only bigBranch nodes that satisfy the predicate. The predicate is the part of the XPath statement that is in square brackets. In this case, the predicate is asking for bigBranch nodes with their ‘name’ attribute set to ‘bb3’.

The last two examples assume we want to specify the path from the root. Let’s now assume that we are specifying the path from a `<smallBranch>` node.

Selection 3:

**English:** The parent node of the current `<smallBranch>`. (Notice that this selection is relative to a `<smallBranch>`)

**Abbreviated:** `..`

**Unabbreviated:** `parent::node()`

When using the Unabbreviated Syntax, you may notice that you are calling a parent or child followed by two colons (::). Each of those are called an axis. You will learn more about axes shortly.

Also, this may be a good time to explain the concept of a location path. A location path is the series of location steps taken to reach the node/nodes being selected. Location steps are the parts of XPath statements separated by / characters. They are one step on the way to finding the nodes you would like to select.

Location steps are comprised of three parts: an axis (child, parents, descendant, etc.), a node test (name of a node, or a function that retrieves one or more nodes), and a series of predicates (tests on the retrieved nodes that narrow the results, eliminating nodes that do not pass the predicate’s test).

So, in a location path, each of its location steps returns a node-list. If there are further steps on the path after a location step, the next step is executed on all the nodes returned by that step.

**Relative vs. Absolute paths**

When specifying a path with XPath, there are times when you will already be ‘in’ a node. But other times, you will want to select nodes starting from the root node. XPath lets you do both. If you have ever worked with websites in HTML, it works the same way as referring to other files in HTML hyperlinks. In HTML, you can specify an Absolute Path for the hyperlink, describing where another page is with the server name, folders, and filename all in the URL. Or, if you are referring to another file on the same site, you need not enter the server name or all of the path information. This is called a Relative Path. The concept can be applied similarly in XPath.

You can tell the difference by whether there is a '/' character at the beginning of the XPath expression. If so, the path is being specified from the root, which makes it an Absolute Path. But if there is no '/' at the beginning of the path, you are specifying a Relative Path, which describes where the other nodes are relative to the context node, or the node for which the next step is being taken.

Below is an XSL stylesheet (Exhibit 9.3) for use with our tree.xml file above (Exhibit 9.2).
Exhibit 9.3: xsl_tree.xsl – Example of both a relative and absolute path

Four types of XPath location paths

In the last two sections you learned about two different distinctions to separate out different location paths: Unabbreviated vs. Abbreviated and Relative vs. Absolute. Combining these two concepts could be helpful when talking about XPath location paths. Not to mention, it could make you sound really smart in front of your friends when you say things like:

1. Abbreviated Relative Location Paths- Use of abbreviated syntax while specifying a relative path.
2. Abbreviated Absolute Location Paths- Use of abbreviated syntax while specifying an absolute path.
3. Unabbreviated Relative Location Paths- Use of unabbreviated syntax while specifying a relative path.
4. Unabbreviated Absolute Location Paths- Use of unabbreviated syntax while specifying an absolute path.

I only mention this four-way distinction now because it could come in handy while reading the specification, or other texts on the subject.

XPath axes

In XPath, there are some node selections whose performance requires the Unabbreviated Syntax. In this case, you will be using an axis to specify each location step on your way through the location path.

From any node in the tree, there are 13 axes along which you can step. They are as follows:

<table>
<thead>
<tr>
<th>Axes</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor::</td>
<td>Parents of the current node up to the root node</td>
</tr>
<tr>
<td>ancestor-or-self::</td>
<td>Parents of the current node up to the root node and the current node</td>
</tr>
<tr>
<td>attribute::</td>
<td>Attributes of the current node</td>
</tr>
<tr>
<td>child::</td>
<td>Immediate children of the current node</td>
</tr>
<tr>
<td>descendant::</td>
<td>Children of the current node (including children's children)</td>
</tr>
<tr>
<td>descendant-or-self::</td>
<td>Children of the current node (including children's children) and the current node</td>
</tr>
<tr>
<td>following::</td>
<td>Nodes after the current node (excluding children)</td>
</tr>
<tr>
<td>following-sibling::</td>
<td>Nodes after the current node (excluding children) at the same level</td>
</tr>
<tr>
<td>namespace::</td>
<td>XML namespace of the current node</td>
</tr>
<tr>
<td>parent::</td>
<td>Immediate parent of the current node</td>
</tr>
<tr>
<td>preceding::</td>
<td>Nodes before the current node (excluding children)</td>
</tr>
<tr>
<td>preceding-sibling::</td>
<td>Nodes before the current node (excluding children) at the same level</td>
</tr>
<tr>
<td>self::</td>
<td>The current node</td>
</tr>
</tbody>
</table>

XPath predicates and functions

Sometimes, you may want to use a predicate in an XPath Location Path to further filter your selection. Normally, you would get a set of nodes from a location path. A predicate is a small expression that gets evaluated for each node in a set of nodes. If the expression evaluates to ‘false’, then the node is not included in the selection.

An example is as follows:

```
//p[@class='alert']
```

In the preceding example, every `<p>` tag in the document is checked to see if its `class` attribute is set to ‘alert’. Only those `<p>` tags with a `class` attribute with value ‘alert’ are included in the set of nodes for this location path.

The following example uses a function, which can be used in a predicate to get information about the context node.

```
/book/chapter[position()=3]
```

This previous example selects only the chapter of the book in the third position. So, for something to be returned, the current `<book>` element must have at least 3 `<chapter>` elements.

Also notice that the position function returns an integer. There are many functions in the XPath specification. For a complete list, see the W3C specification at http://www.w3.org/TR/xpath#corelib

Here are a few more functions that may be helpful:

- `number last()` – last node in the current node set
- `number position()` – position of the context node being tested
- `number count(node-set)` – the number of nodes in a node-set
- `boolean starts-with(string, string)` – returns true if the first argument starts with the second
- `boolean contains(string, string)` – returns true if the first argument contains the second
- `number sum(node-set)` – the sum of the numeric values of the nodes in the node-set
- `number floor(number)` – the number, rounded down to the nearest integer
Example

The following XML document, XSD schemas, and XSL stylesheets examples are to help you put everything you have learned in this chapter together using real life data. As you study this example you will notice how XPath can be used in the stylesheet to call and modify the output of specific information from the document.

Below is an XML document (Exhibit 9.4)
Exhibit 9.4: movies_xpath.xml

Below is the second XML document (Exhibit 9.5)

```xml
<?xml version="1.0" encoding="UTF-8"?>
cities
ten xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="cities.xsd">
city
<cityID>c2</cityID>
cityName>Mandal</cityName>
cityPopulation>13840</cityPopulation>
cityCountry>Norway</cityCountry>
tourismDescription>A small town with a big atmosphere. Mandal provides comfort away from normal luxuries.</tourismDescription>
capitalCity>
cityID>c3</cityID>
cityName>Oslo</cityName>
cityPopulation>533050</cityPopulation>
cityCountry>Norway</cityCountry>
tourismDescription>Oslo is the capital of Norway for many reasons. It is also the capital location for tourism. The culture, shopping, and attractions can all be experienced in Oslo. Just remember to bring your wallet.</tourismDescription>
capitalCity>
cities>
```

Exhibit 9.5: cites__xpath.xml

Below is the Movies schema (Exhibit 9.6)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
<!–Movie Collection–>
<xsd:element name="movieCollection">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="movie" type="movieDetails" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<!–This contains the movie details–>
<xsd:complexType name="movieDetails">
  <xsd:sequence>
    <xsd:element name="movieTitle" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
<!–The contains the genre details–>
<xsd:complexType name="roleDetails">
  <xsd:sequence>
    <xsd:element name="roleIDREF" type="xsd:IDREF"/>
    <xsd:element name="roleType" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:simpleType name="ssnType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}‐\d{2}‐\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="castDetails">
  <xsd:sequence>
    <xsd:element name="castMemberID" type="xsd:ID"/>
    <xsd:element name="castFirstName" type="xsd:string"/>
    <xsd:element name="castLastName" type="xsd:string"/>
    <xsd:element name="castSSN" type="ssnType"/>
    <xsd:element name="castGender" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```
Exhibit 9.6: movies.xsd

Below is the Cities schema (Exhibit 9.7)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:element
      name="cities">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element
            name="city"
            type="cityType"
            maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType
      name="cityType">
    <xsd:sequence>
      <xsd:element
          name="cityID"
          type="xsd:ID"/>
      <xsd:element
          name="cityName"
          type="xsd:string"/>
      <xsd:element
          name="cityPopulation"
          type="xsd:integer"/>
      <xsd:element
          name="cityCountry"
          type="xsd:string"/>
      <xsd:element
          name="tourismDescription"
          type="xsd:string"/>
      <xsd:element
          name="capitalCity"
          type="xsd:IDREF"
          minOccurs="0"
          maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Exhibit 9.7: cities.xsd

Below is the XSL stylesheet (Exhibit 9.8)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key
      name="castList"
      match="castMember"
      use="castMemberID"/>
  <xsl:output
      method="html"/>
  <xsl:template
      match="/">
    <html>
      <head>
        <title>Movie Collection</title>
      </head>
      <body>
        <h2>Movie Collection</h2>
        <xsl:apply-templates
            select="movieCollection"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template
      match="movieCollection">
    <!-- let's say we just want to see the actors. -->
    <xsl:for-each
        select="movie">
      <br/>
      <b><xsl:text>Movie Title: </xsl:text></b>
      <xsl:value-of
          select="movieTitle"/>
      <br/>
      <br/>
      <b><xsl:text>Movie Synopsis: </xsl:text></b>
      <xsl:value-of
          select="movieSynopsis"/>
      <br/>
      <br/>
      <xsl:for-each
          select="movie/role">
        <xsl:sort
            select="key('castList',roleIDREF)/castLastName"/>
        <xsl:number
            value="position()"
            format="\n  0. "/>
        <xsl:value-of
            select="key('castList',roleIDREF)/castFirstName"/>
        <xsl:text>, </xsl:text>
        <xsl:value-of
            select="key('castList',roleIDREF)/castLastName"/>
        <xsl:text>, </xsl:text>
        <xsl:value-of
            select="roleType"/>
        <br/>
        <br/>
      </xsl:for-each>
    </xsl:for-each>
    <!-- calling the variables -->
    <font color="red">
      <p><b>Travel Advertisement</b></p>
      <p><xsl:value-of
          select="$city"/>
      , <xsl:value-of
          select="$country"/>
      </p>
    </font>
  </xsl:template>
</xsl:stylesheet>
```
Summary

Throughout the chapter we have learned many of the features and capabilities of the XML Path Language. You should now have a good understanding of node relationships though the use of the XML tree structure. Using the concept of Abbreviated and Unabbreviated location paths allows us to narrow our searches down to only a particular element by satisfying the predicate in the square brackets. Relative and Absolute are used for specifying the path to your location. The Relative path gives the file location in relation to the current working directory while the Absolute path gives an exact location of a file or directory name within a computer or file system. Both of these concepts can be combined to come up with four types of XPath location paths: Abbreviated Relative, Abbreviated Absolute, Unabbreviated Relative, and lastly Unabbreviated Absolute. If further filtering is required XPath predicates and functions can be used. These allow for the predicate to be evaluated for such things as true/false and count functions. When used correctly XPath can be a very powerful tool in the XML language.

Exercises

Answers

Introduction

Through the use of Uniform Resource Identifiers (URIs), an XLink allows elements to be inserted into XML documents that create links between resources such as documents, images, files and other pages. An XLink is similar in concept to an HTML hyperlink, but is more powerful and flexible.

This chapter will be a general overview of the XLink syntax. It will also provide exposure to some of XLink's basic concepts. For the full XLink specification, see the latest version of the standard at:

http://www.w3.org/TR/xlink

XLink

XLinks create a linking relationship between two or more resources. They allow for any XML element, image, text or markup files to be specified in the link.

By using a method similar to the centralized formatting of XSL stylesheets, XLinks allow a document's hyperlinks to be isolated and centralized in a separate document. As a linked document's addresses changes, the XLink remains functional.

The use of XLink requires the declaration of the XLink namespace (http://www.w3.org/1999/xlink/). This namespace provides the global attributes for type, href, role, arcrole, title, show, actuate, label, from and to. The following example would make the prefix xlink available within the tourGuide element.

```
tourGuide
  xlink:href="http://www.w3.org/1999/xlink">
```
**XLink global attributes**

The following table outlines the attributes that can be used with the xlink namespace. The global attributes are type, href, role, arcrole, title, show, actuate, label, from, and to. The table also includes descriptions of how the attributes can be used.

**Exhibit 1: Table of global attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description and Valid Values</th>
</tr>
</thead>
</table>
| **type**   | Describes the meaning of an item  
  - simple - basic format similar to html linkage  
  - extended - more complex than simple types with multi functional format  
  - resource - provides local resources  
  - locator - provides remote resources  
  - arc - provides the ability to traverse from one resource to another  
  - title - readable name or explanation of link |
| **href**   | Location of resource  
  - value is URI |
| **role**   | Description of XLink's content  
  - value is URI  
  - describes the element whose role it is |
| **arcrole**| Description of XLink's content  
  - value is URI  
  - describe the relationship between the two sides of the arc |
| **title**  | Name displayed, usually short description of link |
| **show**   | Describes behavior of the browser once the XLink has been actuated and loaded  
  - new - load in a new window or frame  
  - replace - load in same window or frame  
  - embed - replace the current item  
  - other - look for information elsewhere  
  - none - not specified |
| **actuate**| Specifies when resource is retrieved or link processing occurs  
  - onRequest - when user requests the link  
  - onLoad - when page is loaded  
  - other - looks for information elsewhere  
  - none - not specified |
| **label, from & to** | Specifies link direction |

**XML schema**
The following XML schema defines a tour guide that contains at least one city. Each city contains one or more attractions. The name of each attraction is an XLink.

**Exhibit 2: XML schema for TourGuide**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!‐‐
Document   : TourGuide.xsd
Created on : February 28, 2006
Author     : Billy Timmins
‐‐>
<!‐‐
Declaration of usage of xlink Namespace
‐‐>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <xsd:element
name="tourGuide">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element
name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

This section will contain the City details

```xml
<!‐‐
<xsd:complexType
name="cityDetails">
  <xsd:sequence>
    <xsd:element
name="cityName" type="xsd:string"/>
    <xsd:element
name="adminUnit" type="xsd:string"/>
    <xsd:element
name="country" type="xsd:string"/>
    <xsd:element
name="continent" type="xsd:string">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Asia"/>
          <xsd:enumeration value="Africa"/>
          <xsd:enumeration value="Australia"/>
          <xsd:enumeration value="Europe"/>
          <xsd:enumeration value="North America"/>
          <xsd:enumeration value="South America"/>
          <xsd:enumeration value="Antarctica"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element
name="population" type="xsd:integer"/>
    <xsd:element
name="description" type="xsd:string"/>
    <xsd:element
name="attraction" type="attractionDetails" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

Note use of xlink

```xml
<xsd:complexType
name="attractionDetails">
  <xsd:sequence>
    <xsd:element
name="attractionName" xlink:type="simple"/>
    <xsd:element
name="attractionDescription" type="xsd:string"/>
    <xsd:element
name="attractionRating" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

**XML document**

The following XML document shows how the XLink, attractionName, defined in the XML schema, is used in an XML document. Note that it is necessary to include xlink:href=”" within the attribute tags in order to define the linked website.

**Exhibit 3: XML document for TourGuide.xsd (using XLink)**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!‐‐
Document   : SomeTourGuide.xml
Created on : February 28, 2006
Author     : Billy Timmins
‐‐>
<!‐‐
Declaration of usage of XLink Namespace
‐‐>
<?xml‐stylesheet href="TourGuide.xsl" type="text/xsl"?>
<tourGuide
xmlns:xsi="http://www.w3.org/2001/XMLSchema‐instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:noNamespaceSchemaLocation="TourGuide.xsd">
  <city>
    <cityName>Atlanta</cityName>
    <adminUnit>Georgia</adminUnit>
    <country>USA</country>
    <continent>North America</continent>
    <population>425000</population>
    <description>Atlanta is the capital of and largest city in the U.S. state of Georgia. </description>
    <attraction>
      <attractionName xlink:href="http://www.georgiaaquarium.org/">Georgia Aquarium </attractionName>
      <attractionDescription>World's Largest Aquarium</attractionDescription>
      <attractionRating>5</attractionRating>
    </attraction>
    <attraction>
      <attractionName xlink:href="http://www.high.org/">High Museum of Art </attractionName>
      <attractionDescription>The High Museum of Art, founded in 1985 as the Atlanta Art Association, is the leading art museum in the Southeastern United States.</attractionDescription>
      <attractionRating>4</attractionRating>
    </attraction>
  </city>
</tourGuide>
```
Underground Atlanta

Go beneath the streets of a bustling downtown, to the heart of a great American city. Underground Atlanta is at the center of it all.

Bush Gardens

The nation’s fourth largest zoo, Bush Gardens is where you can see African animals roaming free and an exciting amusement park featuring its world-famous rides like Kumba and the new inverted roller-coaster, Montu.

Henry B. Plant Museum

Discover a museum which transports you to turn-of-the-century Florida.

XML stylesheet

The following XML stylesheet displays the contents of the XML document.

Exhibit 4: XML stylesheet TourGuide
Summary

XLink is an extremely versatile specification that standardizes the process for linking to other data sources. Not only does XLink support unidirectional linking similar to an anchor tag in HTML but also can be used to create bidirectional links. Additionally, XLink allows for the linkage from any XML element. This gives great freedom to the developer.

Exercises

Answers

Cascading Stylesheets (CSS)

Learning objectives

Upon completion of this chapter, for CSS you will be able to

- know the benefits of using CSS
- know the limitations of CSS, so you are able to find the best solution for your document
- know how to implement and use CSS on an XML document

Introduction

CSS (Cascading Style Sheets) is a language that describes the presentation form of a structured document.

An XML or an HTML based document does not have a set style, but it consists of structured text without style information. How the document will look when printed on paper and viewed in a browser or maybe a cellphone is determined by a style sheet. A good way of making a document look consistent and easy to update is by using CSS, which Wikipedia is a good example of.

History of CSS

Style sheets have been around in one form or another since the beginnings of HTML in the early 1990s. Various browsers included their own style language which could be used to customize the appearance of web documents. Originally, style sheets were targeted towards the end-user; early revisions of HTML did not provide many facilities for presentational attributes, so it was often up to the user to decide how web documents would appear.

As the HTML language grew, however, it came to encompass a wider variety of stylistic capabilities to meet the demands of web developers. With these capabilities, style sheets became less important, and an external language for the purposes of defining style attributes was not widely accepted until the development of CSS.

The concept of Cascading Style Sheets was originally proposed in 1994 by Håkon Wium Lie (http://en.wikipedia.org/wiki/H%E5kon_Wium_Lie). Bert Bos (http://en.wikipedia.org/wiki/Bert_Bos) was at the time working on a browser called Argo which used its own style sheets; the two decided to work together to develop CSS.
A number of other style sheet languages had already been proposed, but CSS was the first to incorporate the idea of "cascading" -- the capability for a document's style to be inherited from more than one "style sheet." This permitted a user's preferred style to override the site author's specified style in some areas, while inheriting, or "cascading" the author's style in other areas. The capability to cascade in this way permits both users and site authors added flexibility and control; it permitted a mixture of stylistic preferences.

Håkon's proposal was presented at the "Mosaic and the Web" conference in Chicago in 1994, and again with Bert Bos in 1995. Around this time, the World Wide Web Consortium was being established; the W3C took an interest in the development of CSS, and organized a workshop toward that end. Håkon and Bert were the primary technical staff on the project, with additional members, including Thomas Reardon of Microsoft, participating as well. By the end of 1996, CSS was nearly ready to become official. The CSS level 1 Recommendation was published in December 1996.

Early in 1997, CSS was assigned its own working group within the W3C. The group began tackling issues that had not been addressed with CSS level 1, resulting in the creation of CSS level 2, which was published as an official Recommendation in May 1998. CSS level 3 is still under development as of 2005.

Why use CSS?

Cleaner Looking Code

A mass of HTML tags which manage design elements generally obscure the content of a page, making the code harder to read and maintain. Using CSS, the content of the page is separated from the design, making content production in formats such as HTML, XHTML, and XML as easy as possible.

Pages Will Load Faster

Non-CSS design typically consists of more code than a CSS-designed website.

In a non-CSS design, the information about the design is reloaded every time a visitor accesses a new page. Additionally, the finer points of design are executed awkwardly. For example, a common method of defining the spacing of a web page is to use blank GIF images inside tables.

Using CSS keeps content and design separated, so much less code will be needed. The CSS file loads only once per session, and is saved locally in the user's cache. All information about dimensions is defined in this stylesheet, rendering awkward constructions like blank GIF images unnecessary.

Although an increasing amount of Internet users have broadband, the size of a web page can be important to users who are limited to dial-up connections. Suppose a dial-up user accesses a company's website, and this visitor experiences lengthy loading times. It is quite possible that the visitor would stop their visit or form an opinion of this company as "slow." In this way, a seemingly small difference could mean added revenue.

Furthermore, bandwidth is not free and most webhosting firms limit the amount used. In fact, many hosts charge based on bandwidth usage, so less code could also reduce costs.

Redesign Becomes Trivial

When used properly, CSS is a very powerful tool that gives a web architect complete control over a site's presentation. It is a notation in which the rules of a design are governed. This becomes very useful for a large website which requires a consistent appearance for every type of element (such as a title, a subtitle, a piece of code, or a paragraph).

For example, suppose a company has a 1,200 page website which took many months to complete. The company then undergoes a rebranding and thus the font, the background, the style of hyperlinks, and so forth needs to be updated with the new corporate design. If the site was engineered properly using CSS, this change would be as simple as editing the appropriate lines of a single CSS file (assuming it is an external stylesheet). If CSS is not used, the code that manages the appearance is stored in each of the pages. In order to update the design in this case, each file would have to be updated individually.

Template:Section expansion request

Graceful Degradation

Template:Section expansion request

Accessibility

People with lowered vision or users with special web browsers, e.g. people that are blind, will probably like a CSS designed website better than one not designed using CSS. Because CSS allows you to define the reading order separately from the visual layout it makes it easier for the special web browsers to read the page. Bear in mind that anyone who wears glasses or contact lenses can be considered to have lower vision.

Many designers lock the font size in pixels which prevents the user changing the font size. Good CSS design allows the user to increase or decrease the font size at will making pages more usable. A significant number of web surfers like to use a magnification of 300% or more.

Giving the user the opportunity to change the font size will not make any difference for the normal user, but it can make a difference for people that have lowered vision. Ask yourself the question: who is the website made for? The visitors or the designer?

Websites designed with CSS tend to display better than table-based designs in the web browsers used in PDAs and cellphones. The use of cellphones for browsing will probably continue to increase. A table-based design will make web pages inaccessible to these users.

Be careful with your CSS designs. Misuse of absolute positioning and absolute rather than relative sizes can make your webpages less accessible rather than more accessible. A good table design is better than a bad CSS design.

Better results in search engines

Extensive use of tables confuses the search engines, they can actually get problems separating content from code. The search engine robots start reading on the top of the page, and they want to find out how relevant the webpage is as fast as possible. Again, less code will make it easier for the search engines to find code that's relevant, and it will probably give your webpage a better ranking.

Disadvantages of CSS

Although an increasing amount of Internet users have broadband, the size of a web page can be important to users who are limited to dial-up connections. Suppose a dial-up user accesses a company's website, and this visitor experiences lengthy loading times. It is quite possible that the visitor would stop their visit or form an opinion of this company as "slow." In this way, a seemingly small difference could mean added revenue.

Furthermore, bandwidth is not free and most webhosting firms limit the amount used. In fact, many hosts charge based on bandwidth usage, so less code could also reduce costs.

Redesign Becomes Trivial

When used properly, CSS is a very powerful tool that gives a web architect complete control over a site's presentation. It is a notation in which the rules of a design are governed. This becomes very useful for a large website which requires a consistent appearance for every type of element (such as a title, a subtitle, a piece of code, or a paragraph).

For example, suppose a company has a 1,200 page website which took many months to complete. The company then undergoes a rebranding and thus the font, the background, the style of hyperlinks, and so forth needs to be updated with the new corporate design. If the site was engineered properly using CSS, this change would be as simple as editing the appropriate lines of a single CSS file (assuming it is an external stylesheet). If CSS is not used, the code that manages the appearance is stored in each of the pages. In order to update the design in this case, each file would have to be updated individually.

Template:Section expansion request

Graceful Degradation

Template:Section expansion request

Accessibility

People with lowered vision or users with special web browsers, e.g. people that are blind, will probably like a CSS designed website better than one not designed using CSS. Because CSS allows you to define the reading order separately from the visual layout it makes it easier for the special web browsers to read the page. Bear in mind that anyone who wears glasses or contact lenses can be considered to have lower vision.

Many designers lock the font size in pixels which prevents the user changing the font size. Good CSS design allows the user to increase or decrease the font size at will making pages more usable. A significant number of web surfers like to use a magnification of 300% or more.

Giving the user the opportunity to change the font size will not make any difference for the normal user, but it can make a difference for people that have lowered vision. Ask yourself the question: who is the website made for? The visitors or the designer?

Websites designed with CSS tend to display better than table-based designs in the web browsers used in PDAs and cellphones. The use of cellphones for browsing will probably continue to increase. A table-based design will make web pages inaccessible to these users.

Be careful with your CSS designs. Misuse of absolute positioning and absolute rather than relative sizes can make your webpages less accessible rather than more accessible. A good table design is better than a bad CSS design.

Better results in search engines

Extensive use of tables confuses the search engines, they can actually get problems separating content from code. The search engine robots start reading on the top of the page, and they want to find out how relevant the webpage is as fast as possible. Again, less code will make it easier for the search engines to find code that's relevant, and it will probably give your webpage a better ranking.

Disadvantages of CSS
The use of CSS for styling has few disadvantages. However some browsers, especially older ones, will sometimes present the page incorrectly. When I was gathering information for this chapter it became clear to me that many experts think that formatting XML with CSS is not the future of the web. The main view is that XSL will be the new standard. So make sure you read through the previous chapter of this book one more time. The formatting parts of XSL and CSS will be quite similar. For example, you will be able to use all CSS1 and CSS2 properties and values in XSL with the same meaning as in CSS.

CSS levels

The first CSS specification to become an official W3C Recommendation is CSS level 1, published in December 1996. Among its capabilities is support for:

- **Typeface**/Font properties such as typeface and emphasis
- **Color** of text, backgrounds, and other elements
- **Text attributes** such as spacing between words, letters, and lines of text
- **Alignment (typesetting)**/Alignment of text, images, tables and other elements
- **Margin**, border, padding, and positioning for most elements
- **Unique identification and generic classification of groups of attributes**

The W3C maintains the CSS1 Recommendation (http://www.w3.org/TR/REC-CSS1).

CSS level 2 was developed by the W3C and published as a Recommendation in May 1998. A superset of CSS1, CSS2 includes a number of new capabilities, among them the absolute, relative, and fixed positioning of elements, the concept of media types, support for aural style sheets and bidirectional text, and new font properties such as shadows. The W3C maintains the CSS2 Recommendation (http://www.w3.org/TR/REC-CSS2/).

CSS level 2 revision 1 or CSS 2.1 fixes errors in CSS2, removes poorly-supported features and adds already-implemented browser extensions to the specification. It's currently a Candidate Recommendation (http://www.w3.org/TR/CSS21/).

CSS level 3 is currently under development. The W3C maintains a CSS3 progress report (http://www.w3.org/Style/CSS/current-work#table).

CSS Syntax and Properties

- The section on selectors has moved to CSS Programming/Selectors.
- The section on color has moved to CSS Programming/Color.
- The section on text has moved to CSS Programming/Fonts and Text.
- The section on borders has moved to CSS Programming/Box Model.
- The section on positioning has moved to CSS Programming/Positioning.

The following section contains a list of some of the most common CSS properties. A complete list can be found here (http://www.htmlhelp.com/reference/css/properties.html). The syntax for the use of CSS in an XML document is the same as that for HTML. The difference is in how you link your CSS file to the XML document. To do this you have to write `<xml-stylesheet href="X.css" type="text/css"/>` before the root element of your XML document, where X.css of course is the name of the CSS file.

As mentioned earlier in this chapter, CSS is a set of rules that determines how elements in a document will be shown. The rule has two parts: a selector and a group of one or more declarations surrounded by braces (curly brackets):

```
selector { declaration; ... }
```

The selector is normally the tag you wish to style. Here is an example of a simple rule containing a single declaration:

```
h1 { color: red; }
```

Result: All h1-elements in the document are shown with the text color red.

The general syntax

Rules are usually defined like this:

```
selector { declaration; ...
```

The declaration is formed like this:

```
property: value;
```

Remember that there can be several declarations in one rule. A common mistake is to mix up colons, which separate the property and value of a declaration, and semicolons, which separate declarations. A selector chooses the elements for which the rule applies and the declaration sets the value for the different properties of the elements that are chosen.

Back to our example:

```
h1 { color: red; }
```

In our example:

```
selector is the element h1
declaration color: red
```

The property **color** gets the value **red**

Multiple declarations can be written either on a single line or over several lines, because whitespace collapses:

```
h1 { color:red; background-color:white; }
```

or

```
h1 {
  color:red;
  background-color:white;
}
```
Details of the properties defined by CSS can be found at CSS Programming#CSS1 Properties.

**Summary**

**Cascading Style Sheets** (CSS), are used with webpages to define the view of information saved in HTML or XML. While XML and HTML create and preserve a documents structure, CSS is used to define the appearance and placement of objects within the document as well as its content. All of this information is saved in a separate file, the .css file. In the CSS file are textsize, background color, text types, e.g defined. The placement of pictures and other animations are also defined in the css file. If CSS is used correctly it would make a webpage a lot easier to create and even more important, to maintain. Because you will only have to make changes in the css file to make the whole website change.

![CSS Zen Garden without CSS](file:Csszengarden nocss.png)

![Zen Garden with CSS](file:Csszengarden1 css.png)

**References and useful links**

- CSS Property Reference
- References:
  - http://no.wikipedia.org/ - The Norwegian version of Wiki
- Useful links:
  - W3C has a CSS validator, located at http://jigsaw.w3.org/css-validator/
  - w3cs pages about css (http://www.w3.org/Style/CSS/)
  - HTMLDog CSS Beginners Guide (http://www.htmldog.com/guides/cssbeginner/)
  - CSS Zen Garden (http://www.csszengarden.com) -- See whats possible with CSS
  - W3C has a CSS validator, located at http://jigsaw.w3.org/css-validator/
  - Web Design Update (http://www.d.umn.edu/itss/support/Training/Online/webdesign/webdev_listserv.html) - A a plain text email digest newsletter. It typically goes out once a week and has a section on CSS. All web designers and developers are invited to join.
  - - Cascading Style Sheets Books (http://www.d.umn.edu/itss/support/Training/Online/webdesign/books.html#css)

**Exercises**

**Exercise 1**

Using the CSS file provided below, create a price list for books as an XML document. ```xml version="1.0"?> Exercise1.css:
```

```xml
<book> Lord of the rings</book>
<isbn>1.000.56439 </isbn>
<title> The Two Towers </title>
<author> J.R.R. Tolkien </author>
<publisher> Penguin </publisher>
<price> 48 EUR </price>
```

**Exercise 2**

Create a personal homepage, where you introduce yourself.

The page should contain one header, one footer, and navigation as a list of links.

**Solutions**

Solutions

**CSS Challenges**

Copy and paste the HTML, then take up the challenge to create a stylesheet to match the picture! See Web Design:CSS Challenges
XML - Managing Data Exchange

Chapters
Appendices
Exercises

Related Topics
Computer Science Home
Library and Information Science Home
Markup Languages

Get Involved
To do list
Contributors list
Contributing to Wikibooks

Introduction

Cocoon is a product of the Apache Software Foundation. It is a powerful server heavily based on Java and XML technology. While it does have a command line interface, most users will be able to do everything they need to with it simply through careful editing of a few configuration files, formatted as XML documents. If you want to see some examples of what Cocoon can do, go to http://MIST5730.terry.uga.edu:8080/cocoon/.

Assumptions

This tutorial is set up based on the user having access to an installation of Cocoon on Terry’s Blaze server. If you do not have this access, simply replace file locations and access methods with those provided by your server administrator. Some programs described may be Windows-only; you will need to find out a suitable replacement if you are a Macintosh or Linux user, although these utilities are often included with the operating system. JEdit is a free text editor that can read and save files on an FTP or SFTP server as easily as on a hard disk, and properly manipulate many different types of files, with the proper plugins. It is available for Windows, Macintosh, some Linux distributions and as a platform-independent Java application at http://www.jedit.org/.

The Sitemap

The primary Cocoon file to be concerned with is sitemap.xmap, located in the root Cocoon directory. It uses XML tags to define things such as different ways to present data, the location of important files, identification of browsers, and the most important aspect, pipelines. The default xmap will be fine for our purposes, and we will only need to look at the last few lines of it, where pipeline matches are defined. This section begins at the tag <map:pipeline>. A pipeline match looks like this:

```
<map:pipeline>
  <map:match pattern="test">  
    <map:generate type="file" src="content/test.xml"/>  
    <map:transform type="xslt" src="stylesheets/test.xslt"/>  
    <map:serialize type="html"/>  
  </map:match>
```

Let’s look at what each line does. The first line tells Cocoon to watch for someone browsing to http://blaze.terry.uga.edu:8080/cocoon/otc/test. When this happens, the actions on the next three lines take place. Cocoon will take the information from the file test.xml within the content directory, and apply the stylesheet test.xslt from the stylesheets directory. It formats this result as an html page, as specified on the fourth line. Cocoon can use different serializers to format data as an html or xhtml page, flash object, pdf, or even OpenOffice document. Unlike when working with XML for other purposes, no XSD schema is needed – simply create and populate fields in the XML file as necessary.

Cocoon Forms

Cocoon forms, or CForms, are a way to use XML structure to create validating form field objects and then arrange them in a template for use. The primary advantage of CForms over using HTML forms is that fields can be validated either with built-in functionality or simple XML attributes. There are several elements required for this. A definition XML file, which holds the fields, called "widgets":

```
<fd:field id="email" required="true">
  <fd:label>Email address:</fd:label>
  <fd:datatype base="string"/>
  <fd:validation>
    <fd:email/>
  </fd:validation>
</fd:field>
```

A template XML file calls on these widgets, adding HTML code to help with look and feel:

```
<wbr/>
<ft:widget-label id="email"/>
```
A JavaScript line that controls the flow of data from one file to the next:

```javascript
function registration() {
   var form = eval('form("registration_definition.xml")');
   var viewData = ("username": form.getElementById("name").getValue(),
                "email": form.getElementById("email").getValue(),
                "password": form.getElementById("password").getValue(),
                "age": form.getElementById("age").getValue(),
                "spam": form.getElementById("spam").getValue());
   cocoon.sendPage("registration-success-pipeline", viewData);
}
```

Pipelines in the sitemap that also control flow:

```xml
<map:match pattern="registration">
   <map:call function="registration"/>
</map:match>

<map:match pattern="registration-display-pipeline">
   <map:generate type="js" src="registration_template.xml"/>
   <map:parameter name="locale" value="en-US"/>
</map:transform>

<map:transform src="forms-samples-styling.xml"/>
<map:serialize/>
</map:match>

<map:match pattern="registration-success-pipeline">
   <map:generate type="js" src="registration_success.js"/>
   <map:serialize/>
</map:match>
```

An XSP can be used in this flow in order to pass submissions to a database.

**XSPs**

XSPs function similarly to JSPs and servlets - they are server-side applications that can support many users at once. Unlike JSPs and servlets, XSPs can use XML tags to accomplish much of their functionality, although they can also use Java code between `<xsp:logic>` tags. One good use for XSPs is passing information to a database or recalling and displaying stored data. While JSPs and servlets have to either call a specific database connector or contain all of the code for connecting within them, Cocoon has a configuration file which holds this information, and XSPs just call the name of the database as specified in WEB-INF/cocoon.xconf:

```xml
<esql:pool dbname="esql:pool"/>
```

XSP code to enter data from a form might look like this:

```xml
<esql:execute-query>
   <esql:query>
      INSERT into otc_users (name, email, password, age, spam) values ('<xsp:expr>esc_name</xsp:expr>', '<xsp:request:get-parameter name="password"/>', '<xsp:request:get-parameter name="age"/>', '<xsp:request:get-parameter name="spam"/>
   </esql:query>
</esql:execute-query>
```

**Exercises**

1. Create a basic XML file and accompanying html stylesheet. Upload them into the proper folders (content and stylesheets respectively) on the Blaze server, and write a pipeline match that would enable you to view the XML content with your stylesheet applied in a browser. Files and match pattern should be named after your own name, for example Bob Jones would use “bjoness”.

2. Follow along with the CForms example located at http://cocoon.apache.org/2.1/userdocs/basics/sample.html. Create and implement at least one widget of your own making. You can view this at work by browsing to http://blaze.terry.uga.edu:8080/cocoon/forms/registration.

3. Browse to opt/tomcat5/webapps/cocoon/forms on Blaze. Examine sitemap-modified.xmap to see how the pipelines could be modified to pass CForm data to an XSP. Test.jsp shows how that data could be inserted into or called from a database.

**Appendix - Accessing the Blaze server**

When you have an account set up on the Blaze server, there are several steps you will need to take in order to be able to work with files in the Cocoon directory. Generally, new user accounts are set up with the user’s UGA MyId as the username, and social security number as the password. This password must be changed at the user’s first login, which requires using an SSH client to accomplish. UGA students can download Secure Shell Utilities 3.1 at http://sitesoft.uga.edu/. Two programs are installed by this download, Secure Shell Client and Secure File Transfer Client.

Open the Secure Shell client and click the “Quick Connect” button located near the top of the window. In the resulting window, enter “blaze.terry.uga.edu” as the Host Name, and your specified username as User Name. Port Number should be set to “22”, and Authentication Method should be “Passworded”. Click “Connect”. In the resulting window, enter your given password and click “OK”. You may see a window asking to save the new host key, click “Yes”. You will now be presented with a text box. It will notify you that your password has expired and must be changed. You will need to enter your given password once, hit enter, enter your desired new password, hit enter, and again enter your desired new password and hit enter. Be aware that nothing you type will show up for security purposes, and you will not be able to delete any typos - you'll have to log in and start over if you mess up. This is all we will be using the Secure Shell Client application for; you can click the “Disconnect” button in the row of small buttons at the top of the screen, and then exit the program.

In order to actually access files on the Blaze server, the Secure File Transfer Client is used. Open it and click the “Quick Connect” button located near the top of the window, entering the same Host Name as with the Secure Shell Client, your new password, and make sure the other settings are the same. Click “Connect.” You will be presented with a Windows Explorer-type screen where you can browse through the files on the Blaze server. To access our Cocoon installation go to the “opt” folder, then the “tomcat5” folder, then the “webapps” folder, then the cocoon folder. Most of our work will be done in the “otc” folder within. To download a file for editing, simply highlight it and click the “Download” button in the row of small buttons at the top of the screen. Once you select a download location and click “Download.” You can then open it in your editor of choice. To upload a file to the server, simply do the reverse – click the “Upload” button in the row of small buttons as the top of the screen, select a file to upload, and click “Upload,” which will put the file in the folder you are currently viewing on the Blaze server.
In the earlier chapters we were taught how to create XML files in detail. This involved the development of XML documents, Style sheets and Schema and their validation. In this chapter, we will focus on different approaches for parsing XML files and when to use them.

But first, it is time to refresh what we have learned about parsing.

The Process of Parsing XML files

One goal of the XML format was to enhance raw data formats like plain text by including detailed descriptions of the meaning of the content. Now, in order to be able to read XML files, we use a parser which basically exposes the document’s content through a so-called API (application programming interface). In other words, a client application accesses the content of the XML document through an interface, instead of having to interpret the XML code on its own!

Simple Text Parsing

One way to extract data from an XML document is simple text parsing – browsing all characters in the document and check for a desired pattern:

```xml
<house>
  <value><int>150,000</int></value>
</house>
```

Let’s say we are interested in the value of the house. Using straight text parsing, we would scan the file for the character sequence `<value><int>` and call it the start pattern. Then, we would further scan the document for the end pattern (i.e. `</int></value>`). Finally, we declare the text string in between these two patterns to be the value of the surrounding `<house>` tag.

Why it doesn't work that way

Obviously, this approach is not suitable for extracting information from large and complex XML documents, since we would have to know exactly what the file looks like and where the information needed is located. From a more general point of view, the structure and semantics of an XML file is determined by the makeup of the document, its tags and attributes – hence, we need a device that is able to recognize and understand this structure and can point out any errors in it. Moreover, it has to provide the content of the document through an interface, so that other applications can access it without difficulty. This device is known as an XML parser.

What a parser does

Almost all programs that need to process XML documents use an XML parser to extract the information stored in the XML document in order to avoid any of the difficulties that occur when reading and interpreting raw XML data. The parser usually is a class library (e.g. a set of Java class files) that reads a given document and checks if it is well-formed according to the W3C specification. Then, any client software can use methods of the interface provided by the parser API to access the information the parser retrieved from the XML file.

All in all, the parser shields the user from dealing with the complex details of XML like assembling information distributed over several XML files, checking for well-formedness constraints, and so on.

Parsing: an Example

To illustrate more clearly what parsing an XML file really means, the following example was created which contains information about some cities. It also keeps track of who is on vacation and demonstrates the parsing process with the currently most common parsing methods.

Example: cities.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
cities>
  city vacation="Sam">
    cityName:Atlanta/cityName>
    cityCountry:US/cityCountry>
  </city>
  city vacation="Tom">
    cityName:Cartagena/cityName>
    cityCountry:Col/cityCountry>
  </city>
</cities>
```
Based on the information stored in this XML document, we can easily check who is on vacation and where. The parser will read the file using one of the various techniques presented later in this chapter.

This process is very complicated and prone to errors of all kinds. Luckily, we will never have to write code for it, because there are plenty of free, fully-functional parsers on the Web. All we do is download a parser class library and access the XML document through the interface provided by the parser software. With more recent builds of Java, most parsers do not even have to be downloaded. In other words, we use the functions or methods included in the class library for extracting the information.

Basically, a parser reads the XML document and tries to recognize the structure of the file itself while checking for errors. It simply checks for start/end tags, attributes, namespaces, prefixes, and so on. Then, the client software can access the information derived from this structure using methods provided by the parser software (i.e. the interface).

The best way to learn about the functionality of a parser is to actually use them; therefore, the next section demonstrates the different methods of parsing.

### Parser APIs (Application Programming Interface)

#### Overview

There are two “traditional” approaches that dominate the market right now, an event-based push-model as represented by SAX (Simple API for XML) and a tree-based model using the DOM (document object model) approach.

However, there is a movement towards newer approaches and techniques that try to overcome the flaws inherent in these traditional models – an event-based pull-model and a “cursor model”, such as VTD-XML, which allows us to browse the XML document just like in the tree-based approach, but simpler and easier to use.

#### SAX (Simple API for XML)

**Description**

The push model, typically exemplified by SAX (www.saxproject.org) is the “gold standard” of XML parsing, since it is probably the most complete and accurate method so far. The SAX classes provide an interface between the input streams from which XML documents are read and the client software which receives the data made available by the parser. The parser browses through the whole document and fires events every time it recognizes an XML construct (e.g. it recognizes a start tag and fires an event – the client software is notified and can use this information… or not).

**Evaluation**

The advantage of such a model is that we don’t need to store the whole XML document in memory, since we are only reading one piece of information at a time. If you recall that the XML structure is a set of nodes of various types (like an element node) – parsing the document with a SAX parser means going through each node one at a time. This makes it possible to read even very large XML documents in a memory-efficient way. However, the fact that the parser only provides information about the node currently read also implies that the programmer of the client software is in charge of saving certain information in a separate data structure (e.g. the parents or children of the currently processed node). Moreover, the SAX approach is pretty much read-only, since it is hard to modify the XML structure when we do not have some sort of global view.

In fact, the parser is in control of what is read when. The user can only wait until a certain event has occurred and then use the information stored in the currently processed node.

**Example:** `TGSAXParser.java`

As mentioned before, the best way to fully understand the concept of the parsing process is to actually use it. In the following code sample, the information about the name and country of the cities that people are vacationing in will be displayed. The SAX API that is part of the Xerces parser package was used for the implementation ((Xerces 2 Homepage (http://xml.apache.org/xerces2-j))):

```java
// import the basic SAX API classes
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;

public class TGSAXParser extends DefaultHandler {
    public boolean onVacation = false;

    // what to do when a start-element event was triggered
    public void startElement(String uri, String name, String qName, Attributes atts) {
        // stores the string in the XML file
        String vacationer = atts.getValue("vacation");
        String cityName = atts.getValue("cityName");
        String cityCountry = atts.getValue("cityCountry");

        // if the start tag is "city" set vacationer to true
        if (qName.equals("city") && onVacation && vacationer != null) {
            onVacation = true;
            System.out.print("\n" + vacationer + " is on vacation in ");
        }
        if (qName.equals("cityName") && onVacation) {
        }
        if (qName.equals("cityCountry") && onVacation)
    }
```
In the following code sample, a list of cities with people on vacation is again created but this time with the tree-based approach:

```java
/**
 * This method is used to stop printing information once the element has been read. It will also reset the onVacation variable for the next element.
 */
public void endElement(String uri, String name, String qName)
{
    //reset flag
    if(qName.equals("city"))
    {
        onVacation = false;
    }
}
/**
 * This method is triggered to store and print the values between the XML tags. It will only print those values if onVacation == true.
 */
public void characters(char[] ch, int start, int length)
{
    if(onVacation)
    {
        for(int i = start; i < start + length; i++)
            System.out.print(ch[i]);
    }
}
public static void main(String[] args)
{
    System.out.println("People on vacation in the following cities:");
    try
    {
        // create a SAX parser from the Xerces package
        XMLReader xml = XMLReaderFactory.createXMLReader();
        TGSAXParser handler = new TGSAXParser();
        xml.setContentHandler(handler);
        xml.setErrorHandler(handler);
        FileReader r = new FileReader("cities.xml");
        xml.parse(new InputSource(r));
    }
    catch(SAXException se)
    {
        System.out.println("XML Parsing Error: " + se);
    }
    catch(IOException io)
    {
        System.out.println("File I/O Error: " + io);
    }
}
```

The defaultHandler: As mentioned before, SAX is completely event-driven. Therefore, we need a handler that “listens” to the input stream coming from the input file (cities.xml in this case).

The SAX API provides interface classes, which we have to extend with our own code to read our own specific XML document. In order to include our code in the SAX API, we just have to extend the DefaultHandler interface with our own class and set the content handler to our custom handler class (which consists of three methods: startElement, endElement and characters)

The startElement() and endElement() methods: These methods are invoked whenever the SAX parser finds a start or end tag respectively. The SAX API provides blank stubs for both methods and we have to fill them with code of our own.

In this case, we want our program to do something whenever the vacation attribute is set, so we set a Boolean variable to true whenever we find such an element and process the node by printing out the character sequence in between the start and end tag. The character method is automatically called whenever a startElement and endElement event was triggered, but prints out the character string only if the onVacation attribute is set.

**DOM (Document Object Model)**

**Description**

The other popular approach is the tree-based model as represented by the DOM (document object model, see W3C Recommendation (http://www.w3.org/DOM/DOMTR)). This method actually works similarly to a SAX parser, since it reads the XML document from an input stream by browsing through the file and recognizing XML structures.

This time, instead of returning the content of the document in a series of small fragments, the DOM method maps the XML hierarchy to a DOM tree object that contains everything from the original XML document. Everything from elements, comments, textual information or processing instructions is stored in the tree object as nodes, starting with the document itself as the root node.

Now that all the information we need is stored in memory, we access the data by using methods provided by the parser software to read or modify objects within the tree. This facilitates random access to the content of the XML document and provides the possibility to modify the data it contains or even create new XML files by transforming a DOM back to an XML document.

**Evaluation**

However, the major downside of this approach is that it requires much more memory and is therefore not suitable for situations where large XML files are used. More importantly, it is somewhat more complex than the simplistic SAX method even for small and simple problems.

**Example: MyDOMParser.java**

In the following code sample, a list of cities with people on vacation is again created but this time with the tree-based approach:

```java
// import all necessary DOM API classes
import org.apache.xerces.parsers.*;
import org.apache.xerces.dom.*;
import org.w3c.dom.*;
public class MyDOMParser{
    public static void main(String[] args) {
        System.out.println("People on vacation in the following cities:");
```
/*
 * creates a DOM parser object
 * DOMParser parser = new DOMParser();
 * parser.parse("cities.xml");
 */

// stores the tree object in a variable
org.w3c.dom.Document doc = parser.getDocument();

// returns a list of all city elements in my city list
NodeList list = doc.getElementsByTagName("city");

// now, for every element in the city list, check if the
// "vacation" attribute is set and if yes, print out the
// information about the vacationer.
for(int i = 0, length = list.getLength(); i < length; i++){
  Element city = (Element)list.item(i);
  Attr vacationer = city.getAttributeNode("vacation");
  if(vacationer != null){
    String v = vacationer.getValue();
    System.out.println(v + " is vacationing in ");
  }
}

// grab information about city name and country
// directly from the DOM tree object
ParentNode cityname = (ParentNode) doc.getElementsByTagName("cityName").item(0);
ParentNode country = (ParentNode) doc.getElementsByTagName("cityCountry").item(0);
System.out.println(cityname.getTextContent() + "", " + country.getTextContent());

// for the exercise

try{
  parser.getDocument();
}{
  catch(Exception e) {
    System.out.println(e.getMessage());
  }
}

// Summary

Choosing an API at the beginning of your XML project is a very important decision. Once you decide which one to use, it is easy to try different vendors without having much trouble, but switching to a different API will be a very time-consuming and costly process, since you will have to redesign your whole program code.

The SAX API is a widely accepted and well-working parser that is easy to implement and works especially well with streaming content (e.g. an online XML source). Because it is a read-only API, you would not be able to modify the underlying XML data source. Since it only reads one node at a time, it is very memory-efficient and fast. However, this implies that your application expects the information to be close together and ordered.

If you want to randomly access the entire document at any point of time, then the DOM approach might be a better choice for you. The DOM API is more complex and harder to implement, but gives you full control over the whole document and lets you modify the data, also. However, it reads the whole XML document into memory, so the DOM API is not suitable for projects with very large XML files.

// Exercise

Recommended optional exercise

Use the code sample for the SAX and DOM parser from this chapter and play around with it. You probably want to print out different nodes or add more constraints. This absolutely optional, but will give you an idea of the main differences between SAX and DOM.

// Now for the exercise

  Create a SAX parser to parse the file movies.xml. The output simply needs to come from your IDE, it does not need to be sent onto a webpage.

TO HELP YOU download this (http://arches.uga.edu/~andy311/MovieSAXParser.zip), it provides a structure of the problem so that you can more easily run the app in NetBeans 5.0.

If you’re interested in using Xerces – just download the following file:

  http://www.apache.org/dist/xml/xerces-j/xerces-J-bin.2.8.0.zip

If the above link is dead. Go to http://www.apache.org/dist/xml/xerces-j/ and download the latest zip binary file. It should be in the format of "Xerces-J-bin.#.#.#.zip"

Then put the content into the \lib\ext subfolder of your NetBeans directory and start up NetBeans IDE. Now, the Xerces package is successfully installed on your machine.

// Useful Links

Learning objectives

- Get a brief overview of what XUL is.
- Learn about the basic tag/widget library.
- Create simple, static XUL web pages.
- Add event handlers to a XUL page.

Introduction

XUL (pronounced zool and rhymes with cool), which stands for eXtensible User interface Language, is an XML-based user interface language originally developed for use in the Netscape browser. It is now maintained by Mozilla. It is a part of Mozilla Firefox and many other Mozilla applications, and is available as part of Gecko, the rendering engine developed by Mozilla. In fact, XUL is powerful enough that the entire user interface in the Firefox application is implemented in XUL.

Like HTML, in XUL you can create an interface using a relatively simple markup language, define the appearance with CSS style sheets, and use JavaScript to manipulate behavior. Unlike HTML, however, XUL provides a rich set of user interface widgets to create, for example, menus, toolbars and tabbed panels.

To put it in simple terms, XUL can be used to create lightweight, cross-platform, cross-device user interfaces.

Many applications are developed using features of a specific platform that makes building cross-platform software time-consuming and costly. Some users may want to use an application on technologies other than traditional computers, such as small handheld devices. To date, there have been some cross-platform solutions already developed. Java, for example, was created just for such a purpose. However, creating GUIs with Java is cumbersome at best. Alternatively, XUL has been designed for building portable user interfaces easily and quickly. It is available on most versions of Windows, Mac OS X, Linux and Unix. Yahoo! currently uses XUL and related technologies for its Yahoo! tool bar (a Firefox extension) and Photomail application.

To illustrate XUL’s potential, this chapter will work through a few examples. Potential is the correct word here. The full capabilities of XUL are beyond the scope of this chapter but it is designed to give the reader a first look at the power of XUL. One more thing needs to be noted: you’ll need a Gecko-based browser (such as Firefox or the Mozilla Suite) or XULRunner (http://developer.mozilla.org/en/docs/XULRunner) to work with XUL.

The Basics

XUL is XML, and like all good XML files, a good XUL file begins with the standard XML version declaration. Currently, XUL is using the XML version 1.0.

To make your XUL page look good, you must include a global stylesheet in it. The URI of the default stylesheet is `href = “chrome://global/skin/”`. While you can load as many stylesheets as you like, it is best practice to load the global stylesheet initially. Look at Fig.1. Notice the reference to “chrome”. ‘The chrome is the part of the application window that lies outside of a window's content area. Toolbars, menu bars, progress bars, and window title bars are all examples of elements that are typically part of the chrome.’(1) Chrome is the descriptive term used to name all of the elements in a XUL application. Think of it like the chrome on the outside of a car. It’s what catches your eye. The elements in a XUL file are what you see in the browser window.

All XML documents must have a namespace declaration. The developers of XUL have provided a namespace that shows where they came up with the name XUL. (The reference is from the movie ‘Ghostbusters’ for the uninitiated)
As was stated earlier, a XUL document is used to create user interfaces. UI's are generally not of interactive components such as text boxes, buttons and the like. A XUL document accomplishes this with the use of widgets, which are self-contained components with pre-defined behavior. For example buttons will respond to mouse clicks and menu bars can hold buttons. All the normally accepted actions of GUI components are built in to the widgets. There is already a rich library of predefined widgets, but because this is open source, any one can define a widget or a set of widgets for themselves. The widgets are ‘disconnected’ until they are programmed to work together. This can be done simply with JavaScript or a more complex application can be made using something like C++ or Java. In this chapter we will use JavaScript to illustrate XUL’s uses and potential.

Also, a XUL file should have .xul extension. The Mozilla browser will automatically recognize it and know what to do with it when you click on it. Optionally, an .xml extension could be used but you would have to open the file within the browser.

One more thing needs to be mentioned. There are a few syntax rules to follow and they are:

- All events and attributes must be written in lowercase.
- All strings must be double quoted.
- Every XUL widget must use close tags (either <tag></tag> or <tag/> to be well-formed.
- All attributes must have a value.

A First Example

What better way to start then with the good old ‘Hello World’ example. Open up a text editor (not MS Word) like notepad or TextPad and type in:

```xml
<?xml version="1.0"?>
<window
<title>Hello World Example
orient="vertical"
width="1000"
height="600"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
>
<description>Hello World Example</description>
<description>Hello World</description>
<label>Hello World</label>
</window>
```

Save it anywhere but be sure to give the file the .xul extension. Now just double click on it and it should open in your Mozilla or Netscape browser. You should get ‘Hello World’ three times, one on top of the other. Notice the different ways that ‘Hello World’ was printed: twice from a description tag and once from a label tag. Both <description> and <label> are text related tags. Using the description tag is the only way to write text that is not contents of a ‘value’ attribute. This means that you can write text that isn’t necessarily assigned to a variable. In the second and third examples the text is expressed as an attribute to the tag description or label, respectively. You can see here that the orient attribute in window is set to ‘vertical’. That is why the text is output in a column. Otherwise, if orient was set to ‘horizontal’, all the text would be on one line. Try it.

Now let’s start adding some more interesting elements.

Adding Widgets

As stated earlier, XUL has an existing rich library of elements fondly called widgets. These include buttons, text boxes, progress bars, sliders and a host of other useful items. One good listing is the XUL Programmer's Reference (http://www.mozilla.org/xpfe/xulref/).

Let us take a look at some simple buttons. Enter the following code and place it into a Notepad or other text editor that is not MS Word.

```xml
<?xml version="1.0"?>
<window
<title>Find File-window</title>
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
>
<button id="find-button" label="Find" default="true"/>
<button id="cancel-button" label="Cancel"/>
</window>
```

Save it and give the file the .xul extension. Open a Mozilla or Netscape browser and open the file from the browser. You should see a "find" button and a "cancel button". From here it is possible to add more functionality and build up elaborate interfaces. There has to be some place to put all of these things and like the <body> tag in HTML, the <box> tag in XUL is used to house the widgets. In other words, boxes are containers that encapsulate other elements. There are a number of different <box> types. In this example we’ll use < hbox>, < vbox>, < toolbox> and < tabbox>.

<hbox> and < vbox> are synonymous with the attributes 'orient = "horizontal"' and 'orient = "vertical"', which respectively form the < window> tag. By using these two boxes, discrete sections of the window can have their own orientation. These two elements can hold all of the other elements and can even be nested. The tags < toolbox> and < tabbox> serve special purposes. < toolbox> is used to create tool bars at the top or bottom of the window while < tabbox> sets up a series of tabbed sheets in the window.

Take the XUL Framework from Fig. 1 and replace "... (add elements here)" with a < vbox> tag pair (that's both open and close tags). This will be the outside container for the rest of the elements. Remember, the < vbox> means that elements will be positioned vertically in order of appearance. Add the attribute 'flex="1"'. This will make the menu bar extend all the way across the window.

```xml
<?xml version="1.0"?>
<window
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
>
<body
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
>
<vbox

```
Now on to the <tabbox>. Let's make three different sheets with different elements on them. Put this code in between the <tabbox> tags:

```
items there are actually checkboxes. The first one is checked by default and the second one is not.
```

```
notice the menu item labeled 'Paste' has an attribute: disabled="true". This causes the Paste label to be grayed out in that menu and finally in the View menu the menu 'Copy' have accelerator text labels. In the File menu there is a <menuseparator/> tag. This places a line across the menu that acts as a visual separator. In the Edit menu, 'accesskey' attribute. This attribute underlines the letter and provides the reference for making a hot key for that menu item. Notice in the Edit menu, both 'Cut' and <menupopup> does just it says. It creates the popup menu that occurs when the menu label is clicked. In the popup menu is the list of menu items. Each of these has an
```
First the <menubar> holds all of the menu items (File, Edit ,View). Next there are the three different menu items. Each menu has a set of elements and attributes. The <menupopup> does just it says. It creates the popup menu that occurs when the menu label is clicked. In the popup menu is the list of menu items. Each of these has an 'accesskey' attribute. This attribute underlines the letter and provides the reference for making a hot key for that menu item. Notice in the Edit menu, both 'Cut' and 'Copy' have accelerator text labels. In the File menu there is a <menuseparator/> tag. This places a line across the menu that acts as a visual separator. In the Edit menu, notice the menu item labeled 'Paste' has an attribute: disabled="true". This causes the Paste label to be grayed out in that menu and finally in the View menu the menu items there are actually checkboxes. The first one is checked by default and the second one is not.

Now put a pair each of <toolbox> and <tabbox> tags inside of the <vbox> tags with <toolbox> first. As was said <toolbox> is used to create tool bars so lets add a toolbar similar to the one at the top of the browser.

This is the code so far:

```
<toolbox flex="1">
  (... add elements here)
</toolbox>
```

The 'flex' attribute needs some explanation since it is a primary way of sizing and positioning the elements on a page. Flex is a dynamic way of sizing and positioning widgets in a window. The higher the flex number (1 being highest), the more that widget gets priority sizing and placement over widgets with lower flex settings. All elements have size attributes, such as width and/or height, that can be set to an exact number of pixels but using flex insures the same relative sizing and positioning when resizing a window occurs.

There should now be a menu bar with "File Edit View" in it and they should each expand when you click on them. Let's examine the elements and their attributes more closely to see how they work.

First the <menubar> holds all of the menu items (File, Edit ,View). Next there are the three different menu items. Each menu has a set of elements and attributes. The <menupopup> does just it says. It creates the popup menu that occurs when the menu label is clicked. In the popup menu is the list of menu items. Each of these has an 'accesskey' attribute. This attribute underlines the letter and provides the reference for making a hot key for that menu item. Notice in the Edit menu, both 'Cut' and 'Copy' have accelerator text labels. In the File menu there is a <menuseparator/> tag. This places a line across the menu that acts as a visual separator. In the Edit menu, notice the menu item labeled 'Paste' has an attribute: disabled="true". This causes the Paste label to be grayed out in that menu and finally in the View menu the menu items there are actually checkboxes. The first one is checked by default and the second one is not.

Now on to the <tabbox>. Let's make three different sheets with different elements on them. Put this code in between the <tabbox> tags:

```
<tabs flex="1">
  <tab id="Tab1" label="Sheet1" selected="true"/>
  <tab id="Tab2" label="Sheet2"/>
  <tab id="Tab3" label="Sheet3"/>
</tabs>
```

```
<tabpanel flex="1" id="Tab1Sheet" orient="vertical">
  ... (add elements here)
</tabpanel>
```

```
<tabpanel flex="1" id="Tab2Sheet" orient="vertical">
  ... (add elements here)
</tabpanel>
```

```
<tabpanel flex="1" id="Tab3Sheet" orient="vertical">
  ... (add elements here)
</tabpanel>
```

```
Hey, the slider works (for free).
```

The slider works (for free).

```
Hey, the slider works (for free).
```

```
Hey, the slider works (for free).
```

```
Hey, the slider works (for free).
```
The tabs are first defined with `<tab>`. They are given an id and label. Next, a set of associated panels is created, each with different content. The first one is to show that like HTML style sheets can be applied in line. The second two sheets have component type elements in them. See how the slider works and the progress bar is running on its own.

XUL has a number of types of elements for creating list boxes. A list box displays items in the form of a list. Any item in such a particular list can be selected. XUL provides two types of elements to create lists, a listbox element to create multi-row list boxes, and a menulist element to create drop-down list boxes, as we have already seen.

The simplest list box uses the listbox element for the box itself, and the listitem element for each item. For example, this list box will have four rows, one for each item.

```xml
<listbox>
    <listitem label="Butter Pecan"/>
    <listitem label="Chocolate Chip"/>
    <listitem label="Raspberry Ripple"/>
    <listitem label="Squash Swirl"/>
</listbox>
```

Assigning values to each of the listitems lets the user be able to reference them later using script. This way, other elements can be reference this items to be used for alternative purposes.

All these elements are very nice and easy to put into a window, but by themselves they don't do anything. Now we have to connect things with some other code.

## Adding Event Handlers and Responding to Events

To make things really useful, some type of scripting or application level coding has to be done. In our example, JavaScript will be used to add functionality to the components. This is done in a similar fashion as to scripting with HTML. With HTML, an event handler is associated with an element and some action is initiated when that handler is activated. Most of the handlers used with HTML are also found in XUL, in addition to some unique ones. Scripting can be done in additional lines of code, but a more efficient way is to create a separate file with the needed scripts inside of it. This allows the page to load faster since the rendering engine doesn’t have to decide what to do with the embedded script tags.

That being said, we’ll first add a simple script, in line, as a first example.

Let’s add an ‘onclick’ event handler to fire an alert box when an element is selected. Inside the `<window>` tag add the line beginning with `onclick`:

```xml
<window
    onclick="alert(event.target.tagName); return false;"
    id="findfile-window"
    title="Find files"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
>
(... add elements here)
</window>
```

Now when you click on any element in the window, you created an alert box that pops up telling you the name of the element. One interesting thing to note: When you click on the text enclosed by the description tag the response is undefined but when you click on the text wrapped by the label tag you get the tabName label. This implies that a description tag is not really an element. After playing with the alert box, delete that line and add this inside the opening tag of the ‘Close’ menu item in the ‘File’ menu:

```xml
<oncommand="window.close()"
```

Now when you click on ‘Close’ or use the ‘C’ as a hot key, the entire window will close. The oncommand event handler is actually preferred over onclick because oncommand can handle hot keys and other non-mouse events.

Let’s try one more thing. Add this right after the opening `<window>` tag.

```xml
<script>
  function show()
```

This implies that a description tag is not really an element. After playing with the alert box, delete that line and add this inside the opening tag of the ‘Close’ menu item in the ‘File’ menu:

```xml
<oncommand="window.close()"
```
These two functions first retrieve a reference to the progress meter and the text element using their ids. Then both functions set the style attributes of the progress meter and text element to have a display of 'visible' or 'none' which will do just that: hide or display those two elements. (The tabpanel for the progress meter has to be displayed in order to see these actions)

Now add two buttons that will provide the event to fire these two methods. First, add a new box element to hold the buttons. The width attribute of the box needs to be set otherwise the buttons will be laid out to extend the length of the window.

```xml
<box width="200px">
  <button id="show" label="Show" default="true" oncommand="show();"/>
  <button id="hide" label="Hide" default="true" oncommand="hide();"/>
</box>
</script>
```

**Style Sheets**

Style sheets may be used both for creating themes, as well as modifying elements for a more elaborate user interfaces. XUL uses CSS (Cascading Style Sheets) for this. A style sheet is a file which contains style information for elements. The style sheet makes it possible to apply certain fonts, colors, borders, and size to the elements of your choice. Mozilla applies a default style sheet to each XUL window. So far, this is the style sheet that has been used for all the XUL documents:

```xml
<?xml-stylesheet href="chrome://global/skin/" type="text/css"/>
```

That line gives the XUL document the default chrome://global/skin/ style sheet. In Mozilla, this will be translated as the file global.css, which contains default style information for XUL elements. The file will still show is this line is left out but it will not be as aesthetically pleasing. The style sheet applies theme-specific fonts, colors and borders to make the elements look more suitable. Even though style sheets can provide a better looking file, adding styles cannot always provide a better view. Some CSS properties do not affect the appearance of a widget, such as those that change the size or margins. In XUL, the use of the "flex: attribute should be used instead of using specific sizes. There are other ways that CSS does not apply, and may be to advanced for this tutorial.

Using a style sheet that you perhaps have already made, you just have to insert one extra line of code pointing to the CSS file you have already made.

```xml
<?xml-stylesheet href="chrome://global/skin/" type="text/css"/>
<?xml-stylesheet href="findfile.css" type="text/css"/>
```

This second line of code references the style sheet, and will take over as the default style sheet used for the XUL document. Sometimes it is desired not to have the style that comes with the default CSS file.

**Answers**

**Conclusion**

The examples shown in this chapter merely scratch the surface of XUL’s capabilities. Even though these examples are very simple, one can see how easy it would be to create more complex UI’s with XUL. With a complete set of the standard components such as buttons and text boxes at the programmer’s disposal, the programmer can code anything in XUL that can be coded in HTML. The cross-platform ability of XUL is another bonus but the fact that it doesn’t work with Microsoft’s Internet Explorer may suppress XUL’s widespread use. There is some hope that due to the delay in the development of the next version of IE that XUL may find it’s way into IE, but don’t hold your breath.

**References**

1. 'Configurable Chrome' (http://www.mozilla.org/xpfe/ConfigChromeSpec.html) by Dave Hyatt (hyatt@netscape.com) (Last Modified 4/7/99)
2. XML User Interface Language (XUL) (http://www.mozilla.org/projects/xul) - The Mozilla Organization
3. XulPlanet (http://www.xulplanet.com)
Learning objectives

- To understand web services, and what they can do.
- To know what SOAP is, and what web services use it for.
- To know what Web Services Description Language is, and how to read a WSDL file.
- To know what the Universal Description, Discovery, and Integration (UDDI) standard is.
- To understand how to use Java to connect to a web service.

Web Services Overview

Web Services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service. Web services make use of XML to describe the request and response, and HTTP as its network transport.

The primary difference between a Web Service and a web application relates to collaboration. Web applications are simply business applications which are located or invoked using web protocols. Similarly, Web Services also perform computing functions remotely over a network. However, Web Services use internet protocols with the specific intent of enabling inter operable machine to machine coordination.

Web Services have emerged as a solution to problems associated with distributed computing. Distributed computing is the use of multiple systems to perform a function rather than having a single system perform it. The previous technologies used in distributed computing, primarily Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM), had some limitations. For example, neither has achieved complete platform independence or easy transport over firewalls. Additionally, DCOM is not vendor independent, being a Microsoft product.

Some of the primary needs for a distributed computing standard were:

- Cross-platform support for Business to Business, as well as internal, communication.
- Concurrency with existing Internet infrastructure as much as possible.
- Scalability, both in number and complexity of nodes.
- Internalization.
- Tolerance of failure.
- Vendor independence.
- Suitability for trivial and non-trivial requests.

Over time, business information systems became highly configured and differentiated. This inevitably made system interaction extremely costly and time consuming. Developers began realizing the benefits of standardizing Web Service development. Using web standards seemed to be an intuitive and logical step toward attaining these goals. Web standards already provided a platform independent means for system communication and were readily accepted by information system users.

The end result was the development of Web Services. A Web Service forms a distributed environment, in which objects can be accessed remotely via standardized interfaces. It uses a three-tiered model, defining a service provider, a service consumer, and a service broker. This allows the Web Service to be a loose relationship, so that if a service provider goes down, the broker can always direct consumers to another one. Similarly, there are many brokers, so consumers can always find an available one. For communication, Web Services use open Web standards: TCP/IP, HTTP, and XML based SOAP.

At higher levels technologies such as XAML, XLANG, (transactional support for complex web transactions involving multiple web services) and XKMS (ongoing work by Microsoft and Verisign to support authentication and registration) might be added.

SOAP

Simple Object Access Protocol (SOAP) is a method for sending information to and from Web Services in an extensible format. SOAP can be used to send information or remote procedure calls encoded as XML. Essentially, SOAP serves as a universally accepted method of communication with web services. Businesses adhere to the SOAP conventions in order to simplify the process of interacting with Web Services.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header/>
  <SOAP:Body>
    <!-- SOAP body -->
  </SOAP:Body>
</SOAP:Envelope>
```

A SOAP message contains either a request method for invoking a Web Service, or contains response information to a Web Service request.

Adhering to this layout when developing independent Web Services provides notable benefits to the businesses. Due to the fact that Web Applications are designed to be utilized by a myriad of actors, developers want them to be easily adoptable. Using established and familiar standards of communication ultimately reduces the amount of effort it takes users to effectively interact with a Web Service.
The SOAP envelope is used for defining and organizing the content contained in web service messages. Primarily, the SOAP envelope serves to macerate the specified document will be used for service interaction. It contains an optional SOAP Header and a SOAP Body. Messages are sent in the SOAP body, and the SOAP head is used for sending other information that wouldn't be expected in the body. For example, if the SOAP:actor attribute is present in the SOAP header, it indicates who the recipient of the message should be.

A web service transaction involves a SOAP request and a SOAP response. The example we will be using is a Web Service provided by Weather.gov. The input is latitude, longitude, a start date, how many days of forecast information desired, and the format of the data. The SOAP request will look like this:

```xml
<SOAP-ENV:Envelope

The startDate was left empty because this will automatically get the most recent data. The format data type is not defined because it is defined in the WSDL document.

The response SOAP looks like this:

```xml

SOAP handles data by encoding it on the sender side and decoding it on the receiver side. The data types handled by SOAP are based on the W3C XML Schema specification. Simple types include strings, integers, floats, and doubles, while compound types are made up of primitive types.

Because they are text based, SOAP messages generally have no problem getting through firewalls or other barriers. They are the ideal way to pass information to and from web services.

Service Description - WSDL

Web Service Description Language (WSDL) was created to provide information about how to connect to and query a specific Web Service. This document also adheres to strict formatting and organizational guidelines. However, the methods, parameters, and service information are application specific. Web Services perform different functionality and contain independent information, however they are all organized the same way. By creating a standard organizational architecture for these services, developers can effectively invoke and utilize them with little to no familiarization. To use a web service, a developer can follow the design standards of the WSDL to easily determine all the information and procedures associated with its usage.

Essentially, a WSDL document serves as an instruction for interacting with a Web Service. It contains no application logic, giving the service a level of autonomy. This enables users to effectively interact with the service without having to understand its inner workings.

The following is an example of a WSDL file for a web service that provides a temperature, given a U.S. zip code.
The binding identifies the binding style and protocol for each operation. In this case, it uses Remote Procedure Call style binding, using SOAP.
Let's go through using Java to query the Temperature Web Service we talked about earlier.

Using Java with web services requires some external libraries. For our purposes, however, we will be focusing on using Java to interact with Web Services.

Port Types are abstract collections of operations. In this case, the operation is `getTemp`.

Finally, messages are used by the operations to communicate - in other words, to pass parameters and return values.

From the WSDL file, a consumer should be able to access data in a web service.

For a more detailed analysis of how this particular web service, please visit Weather.gov (http://www.weather.gov/xml/)

Service Discovery - UDDI

You've seen how WSDL can be used to share interface definitions for Web Services, but how do you go about finding a Web Service in the first place? There are countless independent Web Services that are developed and maintained by just as many different organizations. Upon adopting Web Service practices and methodologies, developers sought to foster the involvement and creative reuse of their systems. It soon became apparent that there was a need for an enumerated record of these services and their respective locations. This information would empower developers to leverage the best practices and processes of Web Services quickly and easily. Additionally, having a central reference of current Web Service capabilities enables developers avoid developing redundant applications.

UDDI defines registries in which services can be published and found. The UDDI specification was created by Microsoft, Ariba, and IBM. UDDI defines a data structure and Application Programming Interface (API).

In the three-tier model mentioned before, UDDI is the service broker. Its function is to enable service consumers to find appropriate service providers.

Connecting to UDDI registries using Java can be accomplished through the Java API for XML Registries (JAXR). JAXR creates a layer of abstraction, so that it can be used with UDDI and other types of XML Registries, such as the ebXML Registry and Repository standard.

Using Java With Web Services

To execute a SOAP message, an application must be used to communicate with the service provider. Due to its flexibility, almost any programming language can be used to execute SOAP message. For our purposes, however, we will be focusing on using Java to interact with Web Services.

Using Java with web services requires some external libraries.

- Apache SOAP Toolkit
- Java Mail Framework
- JavaBeans Activation Framework
- Xerces XML parser

Let's go through using Java to query the Temperature Web Service we talked about earlier.

```java
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class TempClient

public static float getTemp (URL url, String zipcode) throws Exception
```
Call call = new Call();

// Service uses standard SOAP encoding
String encodingStyleURI = Constants.URL_SOAP_ENC;
call.setEncodingStyleURI(encodingStyleURI);

// Set service locator parameters
call.setTargetObjectURI("urn:xmethods-Temperature");
call.setMethodName("getTemp");

// Create input parameter vector
Vector params = new Vector();
params.addElement(new Parameter("zipcode", String.class,zipcode, null));
call.setParams(params);

// Invoke the service ....
Response resp = call.invoke(url,"");

// ... and evaluate the response
if (resp.generatedFault())
    throw new Exception();
else
    // Call was successful. Extract response parameter and return result
    Parameter result = resp.getReturnValue();
    Float rate = (Float) result.getValue();
    return rate.floatValue();

// Driver to illustrate service invocation
public static void main(String[] args)
{
    try {
        URL url = new URL("http://services.xmethods.net:80/soap/servlet/rpcrouter");
        String zipcode = "30605";
        float temp = getTemp(url,zipcode);
        System.out.println(temp);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

This Java code effectively hides all the SOAP from the user. It invokes the target object by name and URL, and sets the parameter zipcode. But what does the underlying SOAP Request look like?

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:n="urn:xmethods-Temperature"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <n:getTemp>
            <zipcode xsi:type="xs:string">
                30605
            </zipcode>
        </n:getTemp>
    </soap:Body>
</soap:Envelope>
```

As you see, the SOAP request uses the parameters passed in by the Java Call to fill out the SOAP envelope and direct the message. Similarly, the response comes back into the Java program as 70.0. The response SOAP is also hidden by the Java program.

```xml
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <return xsi:type="xsd:float">70.0</return>
        </ns1:getTempResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here's an additional example of using Java and SOAP to interact with Web Services. This particular Web Service is called the "US Zip Validator" and takes a ZipCode as a parameter, which then returns a corresponding latitude and longitude. When developing applications to interact with Web Services, the first step should be to review the WSDL document.

The WSDL document for this service is located here: http://www.webservicemart.com/uszip.asmx?WSDL

This document will contain all the necessary instructions for interacting with the "US Zip Validator" Web Service.

SOAPClient4XG

This Java class refers to an XML document (SOAPRequest.xml), which is used as the SOAP message. This document should be included in the same project folder as the Java application invoking the service.

After reviewing the "US Zip Validator" WSDL document, it is clear that we would like to invoke the "getTemp" method. This information is contained within the SOAP body and includes the appropriate parameters.

SOAPRequest.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <soap:Body>
    <GetTemp xmlns="urn:mcds:Weather-Temperature">
      <zipcode xsi:type="xsd:string">80245</zipcode>
    </GetTemp>
  </soap:Body>
</soap:Envelope>
```

Following a successful interaction, the Web Service provider will provide a response that is similar in format to the user request. When developing in NetBeans, run this project and examine the subsequent SOAP message response in the Tomcat output window.
Web Services with Netbeans

The Netbeans version used for this explanation is 5.0.

After Netbeans is open, click on the "Runtime" tab on the left pane, then right-click "Web Services" and select "Add Web Service." In the "URL" field, enter the address of the web service WSDL file, in our example above it is "http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl" and click Get Service Description. This will bring up the information of the web service.

Summary

Web services are applications that use XML to communicate with many different systems to perform a task. To facilitate the use of web services, protocols were developed that allow them to be flexible and scalable. SOAP is used to send and define information and WSDL was created to provide information about how to connect to and query a web service. UDDI describes where these web services can be found.

Exercises

Answers

Appendix

References and Links

- Weather.gov (http://www.weather.gov/)
- WebServices.org (http://www.webservices.org/)
- W3C's Web Services Reference (http://www.w3.org/2002/ws/)
- UDDI.org (http://www.uddi.org/)
- XMethods.net (http://www.xmethods.net/)
- Java API for XML Registries (http://java.sun.com/xml/jaxr/)
- Apache SOAP Toolkit (http://ws.apache.org SOAP)
- JavaMail Framework (http://java.sun.com/products/javamail/)
- Xerces Java Parser (http://xml.apache.org/xerces-j/)
- Jasnowski, Mike. Java, XML, and Web Services Bible
- Microsoft Web Services (http://msdn.microsoft.com/webservices/)
- http://www.xml.com/
- http://www.w3schools.com/soap/soap_intro.asp
- http://www.w3.org/TR/2007/REC-soap12-part0-20070427/
- http://www.eeweek.com/article2/0,1895,1589730,00.asp

History

The XMLHttpRequest object enables JavaScript to make HTTP requests to a remote server without the need to reload the page. It was first implemented by Microsoft as an ActiveX object but is now available as a native object within both Mozilla and Apple's Safari browser. Javascript is used to transfer information back to the server in real time where it can be processed by the server, and then returned instantaneously to the user.

Purpose

The main function of the XMLHttpRequest object is that it provides an easy way for webpages to receive updated information from servers without having to refresh the whole webpage. As a result, the webserver's processing load is reduced and the user receives information faster without seeing any interruptions in service.

Future Application

The XMLHttpRequest object has many improvements over the existing data exchange methods. Many developers still rely on Common Gateway Interchange (CGI) for data exchange. Since CGI has no adequate restriction about the format of data, XML usage is relatively pointless from a data exchange standpoint. Utilizing the inherent abilities of XMLHttpRequest will remove the inadequacies originating from the widespread use of CGI. The XMLHttpRequest object provides a
Native XML Database

The term Native XML Database has become popular since 1999, after the company Software AG released the first version of its native XML server Tamino, which included a native XML database. A definition of a native database is that it:

"[d]efines a (logical) model for an XML document and stores and retrieves documents according to that model." (Bourret, 2002)

To model data in XML, two principle approaches are used: Data-centric documents and Document-centric documents.

- Data-centric documents (for data transport) have fairly regular structure, order typically does not matter, and little or no mixed content.
- Document-centric documents (usually for human consumption) have less regular or irregular structure, significant order of the elements, and lots of mixed content.

Examples of Native databases

<table>
<thead>
<tr>
<th>Product</th>
<th>Developer</th>
<th>License</th>
<th>DB Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tamino</td>
<td>Software AG</td>
<td>Commercial</td>
<td>Proprietary. Relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>through ODBC.</td>
</tr>
<tr>
<td>XediX Multimedia</td>
<td>XediX Tera</td>
<td>Commercial</td>
<td>Proprietary</td>
</tr>
<tr>
<td>Solution</td>
<td>Solution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eXist</td>
<td>Wolfgang Meier</td>
<td>Commercial</td>
<td>Relational</td>
</tr>
<tr>
<td>dbXML</td>
<td>dbXML Group</td>
<td>Open Source</td>
<td></td>
</tr>
<tr>
<td>Xindice</td>
<td>Apache Software</td>
<td>Open Source</td>
<td>Proprietary (Model-based)</td>
</tr>
</tbody>
</table>

eXist

eXist ([http://exist.sourceforge.net/](http://exist.sourceforge.net/)) is an Open Source effort to develop a native XML database system, tightly integrated with existing XML development tools like Apache's Cocoon. The database may be easily deployed, running either standalone, inside a servlet engine, or directly embedded in an application.

Some features that are available in eXist and that can be found in most Native XML databases are:

- Schema-less storage - Documents do not have to be associated to schema or document type, meaning they are allowed to be well formed only.
- Collections - A collection plays a similar role to a directory in a file system. When submitting a query the user can choose a distinct part of the collection hierarchy or even all the documents contained in the database.
- Query languages - The most popular query languages supported by Native XML databases are XPath (with extensions for queries over multiple documents) and XQuery.

Relational Databases

Database vendors such as IBM, Microsoft, Oracle, and Sybase have developed tools to assist in converting XML documents into relational tables.
IBM Technology

**DB2 XML Extender** provides access, storage and transformation for XML data through user-defined functions and stored procedure. It offers 2 key storage models: XML Columns and XML Collections.

1. **XML Column**: stores and retrieves entire XML documents as DB2 column data. Use of XML Columns is recommended when XML documents already exist and/or when there is a need to store XML documents in their entity.

2. **XML Collection**: composes XML Documents from a collection of relational tables.

A **data access definition (DAD)** file is used for both XML Column and XML Collection approaches to define the "mapping" between the database tables and the structure of the XML document.

The DAD file defines the XML document tree structure, using the following kinds of nodes:

- **root_node** - Specifies the root element of the document.
- **element_node** - Identifies an element, which can be the root element or a child element.
- **text_node** - Represents the CDATA text of an element.
- **attribute_node** - Represents an attribute of an element.

---

**Oracle**

Oracle's **XML SQL Utility (XSU)** uses a schematic mapping that defines how to map tables and views, including object-relational features, to XML documents. Oracle translates the chain of object references from the database into the hierarchical structure of XML elements.
A corresponding XML document generated from the given object-relational model looks like:

```xml
<?xml version="1.0"?>
<ROWSET>
  <ROW num="1">
    <FIRSTNAME>JOHN</FIRSTNAME>
    <LASTNAME>SMITH</LASTNAME>
    <PHONENO>7061234567</PHONENO>
    <ADDRESS>
      <ZIP>30601</ZIP>
      <CITY>ATHENS</CITY>
      <STREET>123 MAIN STREET</STREET>
    </ADDRESS>
  </ROW>
</ROWSET>
```

XSU can be used for executing queries in a Java environment and retrieve XML from the database.

```java
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;
// class to test XML document generation as String
class testXMLSQL {
  public static void main(String[] args) {
    try {
      // Create the connection
      Connection conn = getConnection("root", ");
      // Create the query class
      OracleXMLQuery qry = new OracleXMLQuery(conn, "SELECT * FROM Customers");
      // Get the XML string
      String str = qry.getXMLString();
      // Print the XML output
      System.out.println("The XML output is:
" + str);
      // Always close the query to get rid of any resources...
      qry.close();
    } catch(SQLException e) {
      System.out.println(e.toString());
    }
  }
  // Get the connection given the user name and password.
  private static Connection getConnection(String username, String password) throws SQLException {
    // register the JDBC driver...
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    // Create the connection using the OCI8 driver
    Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL", username, password);
    return conn;
  }
}
```

Query Languages

XPath

XPath is a language for addressing parts of an XML document, and is the common locator used by both XSLT and XPointer. An XPath expression is a series of location steps separated by “/”. Each step selects a set of nodes that become the current node(s) for the next step. The set of nodes selected by the expression are the nodes remaining after processing each step in order.

XQuery

XQuery is a query language under development by the World Wide Web Consortium (W3C). The ambitious task is to develop the first world standard for querying Web documents. XQuery is a versatile markup language, capable of labeling the information content of diverse data sources including structured and semi-structured documents, databases, and object repositories.


MySQL 5.1

MySQL has a command line utility for executing queries against a MySQL database; it has an option for using XML as their output format. MySQL also allows conversion to XML; more information can be found in Converting MySQL to XML. MySQL allows users to execute any SQL query. mysqldump allows users to specify which tables to dump and to specify a where clause to restrict the rows that are dumped. In its Beta release of MySQL 5.1, several features have been
In order to understand these New functions, we will use the following table:

| CREATE TABLE Customers (doc VARCHAR(100)); |
| INSERT INTO Customers VALUES |
| <person id="1"> |
| <firstname>John</firstname> |
| <lastname>Smith</lastname> |
| <phoneno>123-5678</phoneno> |
| </person> |
| INSERT INTO Customers VALUES |
| <person id="2"> |
| <firstname>Aminata</firstname> |
| <lastname>Cisse</lastname> |
| <phoneno>123-5679</phoneno> |
| </person> |
| INSERT INTO Customers VALUES |
| <person id="3"> |
| <firstname>Lamine</firstname> |
| <lastname>Smith</lastname> |
| <phoneno>123-5680</phoneno> |
| </person> |

XML Functions

MySQL version 5.1 has functions for searching and changing XML documents: ExtractValue() and UpdateXML().

- **EXTRACTVALUE (XML_document, XPath_string);**
  
  This function takes 2 string arguments: The first parameter correspond to the XML_document string, and the 2nd Parameter XPath_string (XPath expression / locator). This will result in the return of the string containing a value from the document.

  ```sql
  mysql> SELECT EXTRACTVALUE(doc,'//firstname') FROM Customers;
  +--------------------------------------------------+
  | EXTRACTVALUE(doc,'//firstname')                   |
  +--------------------------------------------------+
  | John                                              |
  | Aminata                                           |
  | Lamine                                            |
  +--------------------------------------------------+
  3 rows in set (0.01 sec)
  ```

  ```sql
  mysql> SELECT ExtractValue(doc,'/person[@id="3"]/firstname') as fname FROM Customers;
  +-----+
  | fname|
  +-----+
  |      |
  |      |
  | Lamine|
  +-----+
  3 rows in set (0.02 sec)
  ```

- **UPDATEXML (XML_document, XPath_string, new_value);**
  
  This function takes 3 string arguments: The first two parameters are similar to the ones used with extractValue(), XML_document and XPath_string. The third parameter is the new value that will replace the one found. This function will then returns the changed XML.

  ```sql
  mysql> SELECT UpdateXML(doc,'/person[@id="3"]/phoneno', '<phoneno>111-2233<phoneno>') FROM Customers;
  +--------------------------------------------------+
  | UpdateXML(doc,'/person[@id="3"]/phoneno','<phoneno>111-2233<phoneno>') |
  +--------------------------------------------------+
  | <person id="1"> |
  | <firstname>John</firstname> |
  | <lastname>Smith</lastname> |
  | <phoneno>123-5678</phoneno> |
  | </person> |
  | <person id="2"> |
  | <firstname>Aminata</firstname> |
  | <lastname>Cisse</lastname> |
  | <phoneno>123-5679</phoneno> |
  | </person> |
  ```
Installation

Currently (04/05/06) MySQL 5.1 does not come with the installer (Beta Version).

Quick Windows installation.

Details information can be found in the online Manual:


Summary

Exercises

Answers

Learning objectives

Upon completion of this chapter, you will be able to

- Understand SyncML fundamentals and general syntax.
- Understand how and why SyncML is implemented.
- Quickly locate and use SyncML technical specifications.

Introduction

Mobile devices such as PDAs, pagers, mobile phones and laptops are by nature not always connected to a network. Yet these devices contain applications which require information obtained from a network in order to be useful. While most PDAs and mobile phones contain applications such as calendars, task lists, and address books for storing useful information, this information is far less useful when it is static, only available on the device itself. For example, copies of static information will always be dissimilar when changes are made on one copy or the other. Synchronization offers a device the ability to connect to a network in order to update either the information on the device or the information on the network, such that both sets of information are identical and up-to-date.

Given the proliferation of proprietary mobile devices and protocols, as well as the increasing consumer demand for ubiquitous mobile access of information, leading technology companies saw the need to create a standard, universal language for describing the synchronization actions between devices and applications. They formed a consortium to sponsor the SyncML initiative to create this language.

Currently, the SyncML consortium has been adopted and incorporated into the Open Mobile Alliance (http://www.openmobilealliance.org), a larger group of over 300 companies which sponsors many collaborative technology projects and protocols.

What is SyncML?
SyncML or Synchronization Markup Language is an XML-based, industry-standard protocol for synchronizing mobile data across a variety of mobile networks, platforms and devices. SyncML started as an initiative in mid 2000 by major technology companies such as Ericsson, IBM, Palm Inc., Lotus, Matsushita Ltd. (Panasonic), Motorola, Nokia, Openwave, Starfish Software, Psion and Symbian. Their initiative's goals were to create a universal language from the myriad, proprietary, synchronization protocols used by mobile devices and provide a complete set of synchronization functionality for future devices. The consortium released version 1.0 in December 2000. They then implemented new features and resolved issues with the subsequent version releases, finalizing the protocol with version 1.1 in February 2002.

The SyncML protocol is designed with these goals in mind:

- As a common language, any device should be able to synchronize with any SyncML service (a networked data repository).
- Any service speaking SyncML should be able to synchronize with any SyncML-capable device.
- The protocol must address the limitations of mobile devices, specifically with respect to memory storage.
- It must support a variety of transport protocols such as HTTP, SMTP, Bluetooth and others.
- It must deliver common synchronization commands to all devices.
- It builds upon existing web technologies, specifically XML.
- Support asynchronous communication and error-handling, since the Internet has latency.

SyncML consists of client and server commands enclosed within DTD-defined...

SyncML Fundamentals

Vocabulary

Let's begin by defining a vocabulary:

- **Client** - the mobile device, its application and local database.
- **Server** - a remote system communicating to the system database or application.
- **Modifications** - data in fields in a database are changed.
- **Sync** - The client and server exchange SyncML messages with commands.
- **Package** - SyncML DTD conformant XML markup describing requests or actions to be taken by either a SyncML client or server. A package is a collection of actions to be performed.
- **Message** - the smallest unit of SyncML markup. Large packages are broken into separate messages.
- **Mapping** - using an intermediate identifier to tie two pieces of information together. Example: let's say 'green' is '5', and '5' is nice. What is nice? If you said 'green' you are correct. You've just done mapping!

Abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMEI</td>
<td>International Mobile Equipment Identifier</td>
</tr>
<tr>
<td>GUID</td>
<td>Global Unique Identifier</td>
</tr>
<tr>
<td>LUID</td>
<td>Local Unique Identifier</td>
</tr>
</tbody>
</table>

Messages and Packages

SyncML messages are requests from either a client or server to perform some action. The action may be to synchronize data, perform some checks on data, update a status, or handle any errors with these actions. Messages are bundled together as packages, as kind of a to-do list. Messages are a laundry list of requests, and they can be pieced together out of order if sufficient mapping information is given to identify to which package the message belongs.

SyncML is designed this way to accommodate for errors and dropped messages. Should one message be dropped, a syncML client or server will know there is a problem because the mapping cannot be completed. It will then issue a request for the information to be resent. Once the data is received, the updates to the information can proceed.

Structure of a SyncML message

Like SOAP, there are two parts to the SyncML message, a Sync Header `<SyncHdr>` and Sync Body `<SyncBody>`. The header contains meta-information about the request, such as the target database `<Target>` and source database `<Source>` URIs, Authentication information `<Cred>`, the session ID `<SessionID>`, the message ID `<MsgID>`, and SyncML version declaration `<VerDTD>`. The body contains the actual requests, alerts and data.

Addressing

Addressing is done through the `<source>` and `<LocURI>` tags. A server will have a familiar URI like http://www.chris.syncml.org/sync and a client mobile device will have an IMEI identification number like this 30400495959596904.

Mapping

SyncML is based on the idea that clients and servers can have their own way of mapping information in their databases. Therefore, clients and servers must each have their own set of unique identifiers.

- **Locally Unique Identifiers (LUID)** are numbers assigned by the client to a data object in a local database (like a field or a row). They are non-reusable numbers assigned to these objects by the SyncML client.
- **Globally Unique Identifiers (GUID)** are numbers assigned to a data object for use in a remote database. This identifier is assigned by the server.

LUID and GUID numbers only have to be unique if they are being used in a table between two communicating parties. In other words, these numbers are temporary, used for mapping data to tables and only really exist for the complete duration of transactions between client and server.

The server will create a mapping table to tie the LUID and GUID together.
Change Logs

The Server and Client track of changes made to their databases during synchronization through "change logs". SyncML doesn't define the change logs, instead SyncML does require that the changes and corrections be negotiated between client and server through messages. Using change logs, the Client and Server know which fields need to be updated. The implementation of change tracking in the application which will use SyncML is not defined.

Sync Anchors

During Synchronization, the Client and Server need to know which fields to update. If a client/server application is checking the fields prior to updating/modifying them, how then does the client/server keep track of the position of current field in the database? The answer is "by using Sync Anchors".

There are two kinds of Anchors : Last and Next. The 'Last' anchor describes which updates occurred during the last synchronization event. The 'Next' anchor describes the current and future synchronization request. These anchors describe the events from the standpoint of the sending device.

Anchors are sent back and forth from client and server to keep track of what is happening to the database fields and what's going on in overall through the lifetime of the sync operation.

By coordinating Sync Anchors and change logs with the type of Sync that is requested, the server application can determine and track (with change logs) which information is the most up-to-date. For example, it is possible to overwrite 'newer' information - that is information for which there is the most recent time-stamp in the change log - with older information. This could be done by choosing a sync in which the client tells the server to overwrite it's information with client data. This is called a 'refresh sync from client'. The types of synchs are described below.

Syncs

There are seven types of Synchs in the SyncML 1.1 language. The following section describes the types of synchs:

1. **Two-way Sync** - The client and server exchange information about modified data. The client sends the modifications first.
2. **Slow sync** - a two-way sync in which all fields in the database are checked on a field-to-field basis. This type of sync is used for the first sync, or after a synchronization failure.
3. **One-way sync, client only** - the client sends the modified data first. The server accepts and updates the data and does not send its modifications.
4. **Refresh sync from client** - the client sends the entire database to the server. The server does not sync. Rather, the server replaces the target database with the client's database.
5. **One-way sync, server only** - the server sends the modified data first. The client accepts and updates the data and does not send its modifications.
6. **Refresh sync from server** - the server sends all its information from a database to the client, replacing the client's database.
7. **Server alerted sync** - the server remotely commands the client to initiate one of the above sync types with the server. In this way, the server is remotely-controlling the client.

Sync Initiation

Sync Initiation is the process the client and server must go through prior to an actual Synchronization. The first step is for the client and server to speak the same language, exchanging and revealing each other's capabilities (as defined by device, as in amount of memory, and protocol as defined by DTD). The second step is identification of the databases to be synchronized. Next the two must decide on the type of synchronization. The third and final step is authentication. Once this step is completed successfully, the synchronization activities can begin.

Authentication

The SyncML server can send the client a message containing the <Chal> tag in order to represent an authentication challenge to the information the client is attempting to access. The client must then respond, giving the username and password within the <Cred> tag.

SyncML uses MD5 digest access authentication. The Client and Server exchange credentials during the authentication process, returning error codes if the process breaks down at some point. The <Cred> tag is used in the <SyncHdr> for holding the credentials to be used for authentication.

Common SyncML implementations
Nokia was the first company to make a SyncML-enabled phone. It synchronized the calendar database on the phone. SyncML can synchronize to-do lists, calendars, address books, phone-books, pretty much anything an organizer can do. SyncML is capable of much more. It would be appropriate to use SyncML any time there are two disparate, remote applications which need to share the same data.

### SyncML Syntax

#### SyncML Example

**Abbreviated SyncML example**

```xml
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</VerProto>
    <SessionID>104050403</SessionID>
    <MsgID>5</MsgID>
    <Cred>...</Cred>
  </SyncHdr>
  <SyncBody>
    <Status>...</Status>
    <Sync>
      <Target>target database URI</Target>
      <Source>source database URI</Source>
      <Add>datafield and data</Add>
      <Replace>an existing data field with some data</Replace>
    </Sync>
  </SyncBody>
</SyncML>
```

Notice lines {1} and {18} start the SyncML file with the root tags. Next, the SyncHdr is defined by lines {2} and {8}. Further, lines {3,4} define the versioning information, line {5} defines the sessionID to distinguish which unique dialogue is occurring between client and server applications, line {6} shows the MsgID to uniquely identify this set of requests (this entire markup) to be performed by the requested application. Also in the syncHeader are credentials, on line {7}.

The SyncBody begins on line {9}. In this part of the syncML message, device/application status {10}, target/source URIs {12,13}, and requested actions such as the sync itself between lines {11,16}, Add and Replace {14,15} commands are given.

### WBXML and SyncML

WAP Binary XML (WBXML) is a form of XML whereby the XML tags are abbreviated in order to shorten the markup for transmission to mobile devices, which commonly have bandwidth and memory limitations. The XML tags are encoded into a binary shorthand to save space. Let's take a look at an example so that this will make more sense.

The following is WBXML binary code depicting a SyncML message. Notice in the first line there is a the document type definition, represented here in hexadecimal tokens. Can you see what happens to the following string? "//SYNCML//DTD SYNCML 1.1//EN"

```text
wbxml
```

Immediately following this string are the characters '6D 6C 71'. Each of these represent a SyncML tag.

```text
wbxml abbreviations
```

```text
6D = "<SyncML>
6C = "<SyncHdr>
71 = "<VerDTD>
```

```text
wbxml abbreviations (cont.)
```

```text
C3 03 "1" "." "1" 01 represents the beginning of opaque (xml) data
6D this represents the length of this opaque data
"1" "." "1" The characters "1" followed by "." and "1" represents "</VerDTD>"
```

This tells the SyncML processor that this is the beginning of opaque (xml) data this represents the length of this opaque data The characters "1" followed by "." and "1" represents "</VerDTD>"

All together this WBXML code snippet, 6D6C71C303"1.1"01 represents:

```
SyncML header snippet
```

```text
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
  </SyncHdr>
</SyncML>
```

So you can see how using WBXML shorthand would be a more compact means of representing XML, saving bandwidth for mobile devices.

For more information please refer to Ed Dumbill's articles on syncML with WBXML:
SyncML specifications

The best source of information on SyncML is the protocol itself. Visit the Open Mobile Alliance for the SyncML specifications.

Open Mobile Alliance

Download OMA SyncML Specifications and white papers (http://www.openmobilealliance.org/syncml/technology.html) at the Open Mobile Alliance (http://www.openmobilealliance.org). Or check out the SyncML Articles (http://www.openmobilealliance.org/syncml/downloads.html) at the Open Mobile Alliance (http://www.openmobilealliance.org).

SyncML Implementations

Although the SyncML specifications are useful, you still have to implement the protocol in your application. There are a few toolkits and implementations out there that you can use to get a head start.

SyncML Reference Toolkit

The Open Mobile Alliance has released a toolkit written in C to demonstrate SyncML. You can get it here (http://sourceforge.net/projects/syncml-cotoolkit/). If you can read German, you can get a sample application using the toolkit here (http://www.mistern.de/diplom/index.html).

Funambol

Interested in developing SyncML for Java? Check out the open source project Funambol (https://forge.funambol.org/). It offers a Java and C++ SDK that implements the SyncML data synchronization protocol, a Java-based application framework for building SyncML server applications, and a standalone SyncML server.

Summary

Mobile Device Technology is improving and changing at a rapid pace. As US telecommunication companies implement Third generation (3G) WCDMA technology (wide-band code-division multiple access), or wireless broadband, we will begin to see powerful devices emerge on the market. These devices will be able to deliver full color, video, streaming multimedia and a variety of data services such as Multimedia Messaging Service (MMS) through WAP. In that infrastructure is becoming cheaper, these telecommunication companies are starting to shift towards being service providers and media vendors as opposed to communications utilities. Cingular wireless, multimedia messaging and ringtones services are a good example of the shift of their company towards being a media platform. The companies that will survive will be the ones that listen to customers needs and make easy-to-use services. Telecommunications companies can add value to their services by creating custom applications and services that use SyncML for synchronization.

Exercises

1. Visit the Open Mobile Alliance Website, download the pdf of the SyncML v. 1.1 protocol and review it. Reading this reference is a valuable exercise in learning.
2. Answer these questions:
   • What is WBXML and why is it used?
   • How do you foresee SyncML being used in the future?
   • Name a problematic situation whereby SyncML is the best ‘tool’ for the job.

Answers: 2a) WBXML is Wap Binary XML, it is a form of XML whereby the XML tags are abbreviated in order to shorten the markup for transmission to mobile devices, which commonly have bandwidth and memory limitations. 2b) SyncML will likely be used as a general, standard converging mechanism for synchronizing data sets between systems, not just for mobile devices. 2c) A ticket-tracking system called TNT helped to set a web-based open work request management system. The staff running this system would like to have live data from this system on their PDAs, listing open requests. Currently, the PDA database is synced via a docking sync station attached to the staff members’ PCs. Staff members have to download the request list as a CSV file, convert it into a usable PDA database and upload it to the PDA, making it this process cumbersome, prone to error, and always out-of-date. Recommendation: Create a custom app to push live updates to the PDAs using SyncML over Bluetooth/Wireless.

References

What is SVG?

Based on XML, Scalable Vector Graphics (SVG) is an open-standard vector graphics file format and Web development language created by the W3C, and has been designed to be compatible with other W3C standards such as DOM, CSS, XML, XSLT, XSL, SMIL, HTML, and XHTML. SVG enables the creation of dynamically generated, high-quality graphics from real-time data. SVG allows you to design high-resolution graphics that can include elements such as gradients, embedded fonts, transparency, animation, and filter effects.

SVG files are different from raster or bitmap formats, such as GIF and JPEG that have to include every pixel needed to display a graphic. Because of this, GIF and JPEG files tend to be bulky, limited to a single resolution, and consume large amounts of bandwidth. SVG files are significantly smaller than their raster counterparts. Additionally, the use of vectors means SVG graphics retain their resolution at any zoom level. SVG allows you to scale your graphics, use any font, and print your designs, all without compromising resolution. SVG is XML-based and written in plain text, meaning SVG code can be edited with any text editor. Additionally, SVG offers important advantages over bitmap or raster formats such as:

- **Zooming**: Users can magnify their view of an image without negatively affecting the resolution.
- **Text stays text**: Text remains editable and searchable. Additionally, any font may be used.
- **Small file size**: SVG files are typically smaller than other Web-graphic formats and can be downloaded more quickly.
- **Display independence**: SVG images always appear crisp on your screen, no matter the resolution. You will never experience “pixelated” images.
- **Superior color control**: SVG offers a palette of 16 million colors.
- **Interactivity and intelligence**: Since SVG is XML-based, it offers dynamic interactivity that can respond to user actions.

Data-driven graphics

Because it is written in XML, SVG content can be linked to back-end business processes, databases, and other sources of information. SVG documents use existing standards such as Cascading Stylesheets (CSS) and Extensible Stylesheet Language (XSL), enabling graphics to be easily customized. This results in:

- **Reduced maintenance costs**: Because SVG allows image attributes to be changed dynamically, it eliminates the need for numerous image files. SVG allows you to specify rollover states and behaviors via scriptable attributes. Complex navigation buttons, for example, can be created using only one SVG file where normally this would require multiple raster files.
- **Reduced development time**: SVG separates the three elements of traditional Web workflow – content (data), presentation (graphics), and application logic (scripting). With raster files, entire graphics must be completely recreated if changes are made to content.
- **Scalable server solutions**: Both the client and the server can render SVG graphics. Because the “client” can be utilized to render the graphic, SVG can reduce server loads. Client-side rendering can enhance the user-experience by allowing users to “zoom in” on an SVG graphic. Additionally, the server can be used to render the graphic if the client has limited processing resources, such as a PDA or cell phone. Either way the file is rendered, the source content is the same.
- **Easily updated**: SVG separates design from content, allowing easy updates to either.

Interactive graphics

SVG allows you to create Web-based applications, tools, or user interfaces. Additionally, you can incorporate scripting and programming languages such as JavaScript, Java, and Visual Basic. Any SVG element can be used to modify or control any other SVG or HTML element. Because SVG is text based, the text inside graphics can be translated for other languages quickly, which simplifies localization efforts. Additionally, if there is a connection to a database, SVG
Using the CTRL-Key (or Apple-Key) you can drag your mouse to make a rectangle that specifies the cross-section of the area you will zoom to.

Table 1: Features of the Adobe SVG Viewer

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom In</td>
<td>Using the CTRL-Key (or Apple-Key) you can drag your mouse to make a rectangle that specifies the cross-section of the area you will zoom to.</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Zoom Out</td>
<td>This work just like “Zoom In” except you press the CTRL-Key and the SHIFT-Key at the same time.</td>
</tr>
<tr>
<td>Panning</td>
<td>Pressing the ALT-Key and move the mouse cursor while a hand-icon appears.</td>
</tr>
<tr>
<td>Copy SVG</td>
<td>The purpose of the SVG Viewers “Copy SVG” options is for users to be able to cut-and-paste graphics and/or source code into other applications. Using “Copy SVG” developers are able to make a copy of the source code, which can be pasted into any text editor. Also, after selecting “Copy SVG” and switching to a desktop application such as MS Office users are able to choose either to use the Edit/Paste option to produce a snapshot of the SVGs DOM-tree code (this contains the current structure of the dynamic SVG image) or users can use the Edit/Paste Special option to translate the SVG into a Bitmap image. These options are likely to improve and increase as support for SVG improves in other applications.</td>
</tr>
<tr>
<td>View Source</td>
<td>The SVG Viewers “View Source” menu options allow both compressed and uncompressed SVG source code to instantly be viewed as text in a new browser window. This is a very handy option for designers and developers.</td>
</tr>
<tr>
<td>Save SVG as…</td>
<td>This option allows for quickly saving of SVG content to your local computer by popping up a “save SVG as” form that gives you the option to input the name and location of the file. In version 3 of the Adobe SVG Viewer the option of Saving as GZip compressed SVG (.svgz) was added to the ‘save as’ dialog box.</td>
</tr>
</tbody>
</table>

**SMIL**

The Synchronized Multimedia Integration Language (SMIL, pronounced “smile”) enables simple authoring of interactive audiovisual presentations. SMIL is typically used for “rich media”/multimedia presentations which integrate streaming audio and video with images, text or any other media type. SMIL is an easy-to-learn HTML-like language, and many SMIL presentations are written using a simple text-editor. SMIL can be used with XML to enable video and sound when viewing a SVG.

**Attention Microsoft Windows Mozilla users!**

The Seamonkey and Mozilla Firefox browsers have SVG support enabled natively. If desired, the Adobe SVG Viewer plugin will work with Mozilla Firefox, or the Seamonkey browser. [4](http://kb.mozillazine.org/Adobe_SVG_Viewer) Webkit based browsers also have some SVG support natively.

**Native SVG (Firefox)**

The Mozilla SVG implementation is a native SVG implementation. This is as opposed to plug-in SVG viewers such as the Adobe viewer (which is currently the most popular SVG viewer).

Some of the implications of this are:

- Mozilla can handle documents that contain SVG, MathML, XHTML, XUL, etc. all mixed together in the same 'compound' document. This is being made possible by using XML namespaces.
- Mozilla is 'aware' of the SVG content. It can be accessed through the SVG DOM (which is compatible with the XML DOM) and manipulated by Mozilla's script engine.
- Other Mozilla technologies can be used with SVG. XBL coupled with SVG is a particular interesting combination. It can be used to create graphical widgets (I wonder when we'll see the first SVG-based chrome!) or extend Mozilla to recognize other specialized languages such as e.g. CML (chemical markup language). There are samples of these kinds of more advanced usage patterns on [http://croczilla.com/svg/](http://croczilla.com/svg/).

**rsvg-view**

rsvg-view program is a part of the librsvg package[1]. It may be used as the default svg opener. It can resize svgs and export them to png which is often the only thing one needs to do with an svg file.[2]

**Creating SVG files**

**How to do it**

One can use 4 groups of programs:

- general text editors, like Notepad ++ ( with XML syntax highlithning)
- specialized svg editors
- programs that can exports svg ( like gnuplot, Maxima CAS)
- own programs to create svg files directly thru concatenate of strings

**SVG editors**

As you can see from the previous example of a path definition, SVG files are written in an extremely abbreviated format to help minimize file size. However, they can be very difficult to write depending on the complexity of your image. There are SVG editor tools that can help make this task easier. Some of these tools are:
<table>
<thead>
<tr>
<th>SVG Editor</th>
<th>Platform</th>
<th>Availability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sodipodi</td>
<td>Linux / UNIX</td>
<td>Open Source (Free, with source)</td>
<td>Fast vector graphics WYSIWYG editor.</td>
</tr>
<tr>
<td>Adobe Livemotion 2</td>
<td>Win98/ME, Win2000/XP</td>
<td>Commercial product</td>
<td>Adobe Livemotion is the authoring tool similar to Macromedia Flash. It had SVG export capability in earlier version, but in Version 2, its support is withdrawn. It looks that even Adobe's support of the SVG is dubious.</td>
</tr>
<tr>
<td>Beez</td>
<td>Win95/98/ME, WinNT/2000/XP</td>
<td>Free download</td>
<td>Beez is a WYSIWYG editor to create a single animated SVG path, consisting of multiple Bezier curves, which can then be used in an SVG file. Nice utility for hand coders. It is an open-source project, on sourceforge, and written in Delphi.</td>
</tr>
<tr>
<td>Corel Draw!</td>
<td>Win95/98/ME, WinNT/2000, Mac OS X version 11</td>
<td>Commercial product</td>
<td>Has SVG import and export capability</td>
</tr>
<tr>
<td>Gill (Gnome Illustration Application)</td>
<td>Linux / UNIX (with Gnome)</td>
<td>Free, with source</td>
<td>Drawing program with SVG import and export; has a full DOM; continuously updated, can embed SVG in other Gnome programs (such as Gnumeric, the spreadsheet). See the CVS changelog for latest status</td>
</tr>
<tr>
<td>IMS Web Dwarf</td>
<td>Win95/98/ME, WinNT/2000/XP</td>
<td>Free download</td>
<td>WYSIWYG editor, exports to either HTML or SVG</td>
</tr>
<tr>
<td>IMS Web Engine</td>
<td>Win95/98/ME, WinNT/2000/XP</td>
<td>14-day trial downloadable</td>
<td>IMS Web Engine is an Interactive Animation Editor and Web Top publisher for the creation of content rich interactive Dynamic HTML and SVG</td>
</tr>
<tr>
<td>Inkscape (<a href="http://www.inkscape.org/">http://www.inkscape.org/</a>)</td>
<td>Linux, Windows, Mac</td>
<td>Free, with source</td>
<td>WYSIWYG editor, but allows editing the XML directly. No animation yet.</td>
</tr>
</tbody>
</table>

C

Here is example in C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const double PI = 3.1415926535897932384626433832795;
const int Xmax = 1000;
ymax = 1000;
radius=100;
x=200;
y=200;
const char * Black = "#000000", /* hexadecimal number as a string for svg color*/
white = "#FFFFFF";

FILE * fp;

void draw_circle(FILE * filep, int radius, int cx, int cy)
{
fprintf(filep, "<circle cx=""%f"" cy=""%f"" r=""%f"" style=""stroke:%s; stroke-width:2; fill:%s""/>
", cx, cy, radius, white, black);
}

cmd beginSVG(
int main()
{FILE * fp;
char * filename = "circle.svg";
fp = fopen(filename, "w");
```
Haskell


Java Script

Matlab

Based on code by Guillaume JACQUENOT [3]

Lisp

One can use cl-svg library or your own procedure.

Maxima

Python

One can use a prepared library, or wrap the svg code in single quotes.

Getting started
Because SVG is based on XML, SVG follows standard XML conventions. Every SVG file is contained within an ‘svg’ tag as its parent element. SVG can be embedded within a parent document or used independently. For example, the following shows an independent SVG document:

**Exhibit 1: Creating a SVG**

```
<?xml version="1.0" standalone="no"?>
<svg
  width="100%" height="100%" version="1.1"
  xmlns="http://www.w3.org/2000/svg">
  ...
</svg>
```

The first line declares that the code that follows is XML. Note the “standalone” attribute. This denotes that this particular file does not contain enough processing instructions to function alone. In order to attain the required functionality it needs to display a particular image, the SVG file must reference an external document.

The second line provides a reference to the Document Type Definition, or DTD. As mentioned in Chapter 7: XML Schemas, the DTD is an alternate way to define the data contained within an XML instanced document. Developers familiar with HTML will notice the DTD declaration is similar to that of an HTML document, but it is specific for SVG. For more information about DTDs, visit: http://www.w3schools.com/dtd/dtd_intro.asp

**Hint:** Many IDEs (ex. NetBeans) do not have SVG “templates” built in to the tool. Therefore, it may be easier to use a simple text editor when creating SVG documents. Once you have an SVG Viewer installed, you should then be able to open and view your SVG document with any browser. When creating your SVG documents, remember to:

- Declare your document as an XML file
- Make sure your SVG document elements are between <svg> element tags, including the SVG namespace declaration.
- Save your file with a .svg file extension.
- It is not necessary to include a DOCTYPE statement, which includes information to identify this as an SVG document (since SVG 1.2 there is also not more such).[^1][^2][^3][^4]

The <svg> element on line 5 defines the SVG document, and can specify, among other things, the user coordinate system, and various CSS unit specifiers. Just like with XHTML documents, the document element must include a namespace declaration to declare the element as being a member of the relevant namespace (in this case, the SVG namespace). Within the <svg> element, there can be three types of drawing elements: text, shapes, and paths.

**Text**

The following is an example of the text element: **Exhibit 2: Using text with SVG**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
  xmlns="http://www.w3.org/2000/svg"
  width="5.5in" height="0.5in" xml:space="preserve">
  <text
    y="15"
    fill="red">
    This is SVG.
  </text>
</svg>
```

The <svg> element on line 4 specifies: 1) that white space within text elements will be retained, 2) the width and height of the SVG document — particularly important for specifying print output size. In this example, the text is positioned in a 5.5 inches wide by .5 inches tall image area. The “y” attribute on line 5 declares that the text element’s baseline is 15 pixels down from the top of the SVG document. An omitted “x” attribute on a text element implies an x coordinate of 0.

Because SVG documents use a W3C DTD, you can use the W3C Validator to validate your document. Notice that the “style” attribute is used to describe the presentation of the text element. The text could equivalently have been given a red color by use of a presentation attribute fill="red".

**Shapes**

SVG contains the following basic shape elements:

- Rectangles
- Circles
- Ellipses
- Lines
- Polylines
- Polygons

These basic shapes, along with “paths” which are covered later in the chapter, constitute the graphic shapes of SVG. Because this is an introduction to SVG, we will only cover rectangle and circle shapes here. For more information on all shapes, please visit: http://www.w3schools.com/svg/default.asp

**Rectangles**

The <rect> element defines a rectangle which is axis-aligned with the current user coordinate system, the coordinate system that is currently active and which is used to define how coordinates and lengths are located and computed on the current canvas. Rounded rectangles can be created by setting values for the rx and ry attributes.

The following example produces a blue rectangle with its top left corner aligning with the top left corner of the image area. This utilizes the default value of “0” for the x and y attributes.

**Exhibit 3: Creating a rectangle in SVG**

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <rect fill="blue" width="200" height="200"/>
</svg>
```

It will produce this result:
Circles

A circle element requires three attributes: cx, cy, and r. The 'cx' and 'cy' values specify the location of the center of the circle while the 'r' value specifies the radius. If the 'cx' and 'cy' attributes are not specified then the circle's center point is assumed to be (0, 0). If the 'r' attribute is set to zero then the circle will not appear. Unlike 'cx' and 'cy', the 'r' attribute is not optional and must be specified. In addition the keyword stroke creates an outline of the image. Both the width and the color can be changed.

Exhibit 4: Creating a circle in SVG

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" width="350" height="300">
  <circle cx="100" cy="50" r="40" stroke="darkslategrey" stroke-width="2" fill="red"/>
</svg>
```

It will produce this result:

![Circle](image)

Polygons

A polygon is any geometric shape consisting of three or more sides. The 'points' attributes describes the (x,y) coordinates that specify the corners points of the polygon. For this specific example, there are three points which indicate that a triangle will be produced.

Exhibit 5: Creating a Polygon in SVG

```
<?xml version="1.0" standalone="no"?>
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <polygon points="220,100 300,210 170,250" style="fill:#blue;stroke:red;stroke-width:2"/>
</svg>
```

It will produce this result:

![Polygon](image)
Paths

Paths are used to draw your own shapes in SVG, and are described using the following data attributes:

Table 2: SVG Paths

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Command</th>
<th>Parameters</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moveto</td>
<td>M</td>
<td>x y</td>
<td>Set a new current point</td>
<td>Start a new sub-path at the given (x,y) coordinate.</td>
</tr>
<tr>
<td>Lineto</td>
<td>L</td>
<td>x y</td>
<td>Draw a straight line</td>
<td>Draw a line from the current point to the given (x,y) coordinate which becomes the new current point.</td>
</tr>
<tr>
<td>Horizontal lineto</td>
<td>H</td>
<td>x</td>
<td>Draw a horizontal line</td>
<td>Draws a horizontal line from the current point (cpx, cpy) to (x, cpy).</td>
</tr>
<tr>
<td>Vertical lineto</td>
<td>V</td>
<td>y</td>
<td>Draw a vertical line</td>
<td>Draws a vertical line from the current point (cpx, cpy) to (cpx, y).</td>
</tr>
<tr>
<td>Curveto</td>
<td>C</td>
<td>x1 y1 x2 y2 x y</td>
<td>Draw a curve using a cubic Bezier</td>
<td>Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve.</td>
</tr>
<tr>
<td>Smooth curveto</td>
<td>S</td>
<td>x2 y2 x y</td>
<td>Draw a shorthand/smooth curve using a cubic Bezier</td>
<td>Draws a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point. (x2,y2) is the second control point (i.e., the control point at the end of the curve)</td>
</tr>
<tr>
<td>Quadratic Belzier curveto</td>
<td>Q</td>
<td>x1 y1 x y</td>
<td>Draws a quadratic Bézier curve</td>
<td>Draws a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point.</td>
</tr>
<tr>
<td>Smooth quadratic Belzier curveto</td>
<td>T</td>
<td>x y</td>
<td>Draws a shorthand/smooth quadratic Bézier curve</td>
<td>Draws a quadratic Bézier curve from the current point to (x,y).</td>
</tr>
<tr>
<td>Elliptical arc</td>
<td>A</td>
<td>rx ry x-axis-rotation large-arc-flag sweep-flag x y</td>
<td>Draw an elliptical or circular arc</td>
<td>Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (rx, ry) and an x-axis-rotation, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center (cx, cy) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.</td>
</tr>
<tr>
<td>Closepath</td>
<td>Z</td>
<td>(none)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Close the current path by drawing a line to the last moveto point

The following example produces the shape of a triangle. The “M” indicates a “moveto” to set the first point. The “L” indicates “lineto” to draw a line from “M” to the “L” coordinates. The “Z” indicates a “closepath”, which draws a line from the last set of L coordinates back to the M starting point.

Exhibit 6: Creating paths in SVG

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" width="5.5in" height="2in">
  <path d="M 50 10 L 350 10 L 200 120 z"/>
</svg>
```

It produces this result:

![Triangle SVG example](image)

Validation

After creating file check its code with the W3C Validatior[7]

Optimisation

Even code without errors can be improved. For example grouping elements makes code shorter.

Including SVG in HTML

There are three methods to include SVG in an HTML document. Basically, the SVG document is first created as a stand-alone file. It is then referenced in the HTML document using one of the following commands:

Table 4: Including SVG in HTML

<table>
<thead>
<tr>
<th>Command</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| <embed>   | 1. Supported in nearly any browser  
2. Allows html2svg and svg2html scripting  
3. Recommended by Adobe for their SVG Viewer | 1. Not clearly standardized within any HTML specification |
| <object>  | 1. HTML4 and higher standard  
2. Supported in newer browser generations | 1. Works on newer browsers but without html2svg and svg2html scripting |
| <iframe>  | 1. Works in most browsers, but not documented | 1. Generates a window-like border without specific styling |

Embed

The syntax is as follows: Exhibit 7: Embedding SVG into HTML using keyword embed

```
<embed src="canvas.svg" width="350" height="176" type="image/svg+xml" name="emap">
```

An additional attribute, “pluginspage”, can be set to the URL where the plug-in can be downloaded:

```
pluginspage="http://www.adobe.com/svg/viewer/install/main.html"
```

Object

The syntax is as follows and conforms to the HTML 4 Strict specification: Exhibit 8: Embedding SVG into HTML using keyword object

```
<object type="image/svg+xml" usemap="emap" data="canvas_norelief.svg" width="350" height="176"></object>
```

Between the opening and the closing <object> tags, information for browsers that do not support objects can be added:

```
<object ...You should update your browser/object>
```

Subject ...
Unfortunately some browsers such as Netscape Navigator 4 do not show this alternative content if the type attribute has been set to something other than `text/html`.

Iframe

The syntax is as follows and conforms to the HTML 4 Transitional specification: Exhibit 9: Embedding SVG into HTML using keyword iframe

```
<iframe src="canvas_norelief.svg" width="350" height="176" name="imap"></iframe>
```

Between the opening and the closing `<iframe>` tags, information for browsers that do not support iframes can be added:

```
<iframe ...>You should update your browser</iframe>
```

Creating 3D SVG images

Section by Charles Gunti, UGA Master of Internet Technology Program, Class of 2007

Sometime we may want to view an SVG image in three dimensions. For this we will need to change the viewpoint of the graphic. So far we have created two dimensional graphics, such as circles and squares. Those exist on a simple x, y plane. If we want to look at something in three dimensions we have to add the z coordinate plane. The z plane is already there, but we are looking at it straight on, so if data is changed on z it doesn't look any different to the viewer. We need to add another parameter to the data file, the z parameter.

```
<?xml version="1.0">
<data>
  <subject x_axis="90" y_axis="118" z_axis="0" color="red"/>
  <subject x_axis="113" y_axis="45" z_axis="75" color="purple"/>
  <subject x_axis="-30" y_axis="-59" z_axis="110" color="blue"/>
  <subject x_axis="60" y_axis="-50" z_axis="-25" color="yellow"/>
</data>
```

Once we have the data we will use XSLT to create the SVG file. The SVG stylesheet is the same as other stylesheets, but we need to ensure an SVG file is created during the transformation. We call the SVG namespace with this line in the declarations:

```
xmlns="http://www.w3.org/2000/svg"
```

Another change we should make from previous examples is to change the origin of (0, 0). We change the origin in this example because some of our data is negative. The default origin is at the upper left corner of the SVG graphic. Negative values are not displayed because, unlike traditional coordinate planes, negative values are above positive values. To move the origin we simply add a line of code to the stylesheet. Before going over that line, let's look at The g element. The container element, g, is used for grouping related graphics elements. Here, we'll use g to group together our graphical elements and then we can apply the transform. Here is how we declare g and change the origin to a point 300 pixels to the right and 300 pixels down:

```
<g transform="translate(300,300)">graphical elements</g>
```

SVG transformations are pretty simple, until it comes to changing the viewpoint. SVG has features such as rotating and skewing the image in two dimensions, but it cannot rotate the coordinate system in three dimensions. For that we will need to use some math and a little Java. When rotating in three dimensions two rotations need to be made, one around the y axis, and another around the x axis. The first rotation will be around the y axis and the formula will look like this:

$$
Az = z \cdot \cos(-Az) - x \cdot \sin(-Az)
$$

Az is the angle the z axis will be rotated

$$
x' = x \cdot \cos(-Az) - y \cdot \sin(-Az)
$$

y will not change because we are rotating around the y axis

The second rotation will be around the x axis. Keep in mind that one rotation has already been made, so instead of using x, y, and z values we need to use x', y', and z' (x-prime, y-prime and z-prime) found in the last rotation. The formula will look like this:

$$
x'' = z' \cdot \cos(Ay) - y' \cdot \sin(Ay)
$$

Ay is the angle of rotation on the y axis

$$
y'' = z' \cdot \sin(Ay) + y' \cdot \cos(Ay)
$$

x'' = x' Remember we are rotating around the x axis, so this does not change

Remember from trig class the old acronym SOH CAH TOA? This means

Sin = Opposite/Hypotenuse Cos = Adjacent/Hypotenuse Tan = Opposite/Adjacent

And we use those functions to find the angles needed for our rotations. Based of the previous two formulas we can make the following statements about Az and Ay:

$$
\tan(Az) = Xv/Zv
$$

$$
\sin(Ay) = Yv/\sqrt{Xv^2 + Yv^2 + Zv^2}
$$

With so many steps to take to make the rotation we should drop all of this information into a Java class, then call the class in the stylesheet. The Java class should have methods for doing all of the calculations for determining where the new data points will go once the rotation is made. Creating that java class is beyond the scope of this section, but for this example I'll call it ViewCalc.class.

Now that we can rotate the image, we need to integrate that capability into the transformation. We will use parameters to pass viewpoints into the stylesheet during the transformation. The default viewpoint will be (0, 0, 0) and is specified on the stylesheet like so:

```
Exhibit 10: 3D images with SVG
```
Java now needs to be added to the stylesheet so the processor will know what methods to call. Two lines are added to the namespace declarations:

```xml
<xsl:template match="/">
  ...
  Java now needs to be added to the stylesheet so the processor will know what methods to call. Two lines are added to the namespace declarations:
  
  <?xml version="1.0"/>
  <xsl:stylesheet version="1.0" xmlns="http://www.w3.org/2000/svg" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!‐‐ default viewpoint in case they are not specified →>
    <!‐‐ from the command line –‐>
    <xsl:param name="viewpoint_x">0</xsl:param>
    <xsl:param name="viewpoint_y">0</xsl:param>
    <xsl:param name="viewpoint_z">0</xsl:param>
  </xsl:stylesheet>
```

Notice the `exclude-result-prefixes="java"` line. That line is added so things in the stylesheet with the `java:` prefix will be processed, not output. Be sure to have the ViewCalc class in the CLASSPATH or the transformation will not run.

The final step is to call the methods in the ViewCalc class from the stylesheet. For example:

```xml
<xsl:template match="/">
  ...
  Java now needs to be added to the stylesheet so the processor will know what methods to call. Two lines are added to the namespace declarations:
  
  <?xml version="1.0"/>
  <xsl:stylesheet version="1.0" xmlns="http://www.w3.org/2000/svg" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!‐‐ default viewpoint in case they are not specified →>
    <!‐‐ from the command line –‐>
    <xsl:param name="viewpoint_x">0</xsl:param>
    <xsl:param name="viewpoint_y">0</xsl:param>
    <xsl:param name="viewpoint_z">0</xsl:param>
  </xsl:stylesheet>
```

Finally we pass new parameters and run the XSL transformation to create the SVG file with a different viewpoint.

**Summary**

SVG stands for Scalable Vector graphics. Meaning that it creates an image that will not lose image quality when moving or changing the size. Similar to Flash in functionality, neither is better than the other, they are however better in particular situations (some of which were listed earlier.) Can create both 2D and 3D images via SVG. Supported by W3C.

**Demos**

The following table provides a sampling of SVG documents that demonstrate varying degrees of functionality and complexity:

<table>
<thead>
<tr>
<th>Function</th>
<th>URL</th>
<th>Browser Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td><a href="http://www.carto.net/papers/svg/samples/canvas.svg">http://www.carto.net/papers/svg/samples/canvas.svg</a></td>
<td>All</td>
</tr>
<tr>
<td>Fills</td>
<td><a href="http://www.carto.net/papers/svg/samples/fill.svg">http://www.carto.net/papers/svg/samples/fill.svg</a></td>
<td>All</td>
</tr>
<tr>
<td>HTML, JS, Java Servlet</td>
<td><a href="http://www.adobe.com/svg/viewer/install/main.html">http://www.adobe.com/svg/viewer/install/main.html</a></td>
<td>- Then follow to Inspiration, Fluent Solutions/Adobe Theater demo</td>
</tr>
<tr>
<td>HTML, JS, DOM</td>
<td><a href="http://www.adobe.com/svg/viewer/install/main.html">http://www.adobe.com/svg/viewer/install/main.html</a></td>
<td>- Then follow to Inspiration, Chart and Graph demo</td>
</tr>
<tr>
<td>PHP, MySQL</td>
<td><a href="http://www.carto.net/papers/svg/samples/mysql_svg_php.shtml">http://www.carto.net/papers/svg/samples/mysql_svg_php.shtml</a></td>
<td>All</td>
</tr>
</tbody>
</table>

The **Basic** demo demonstrates the effects of zooming, panning, and anti-aliasing (high quality).

The **Fills** demo demonstrates the effects of colors and transparency. The black circle is drag-able. Simply click and drag the circle within the square to see the changes.

The **HTML, JS, Java Servlet** demo describes an interactive, database-driven, seating diagram, where chairs represent available seats for a performance. If the user moves the mouse pointer over a seat, it changes color, and the seat detail (section, row, and seat number) and pricing are displayed. On the client side of the application, SVG renders the seating diagram and works with JavaScript to provide user interactivity. The SVG application is integrated with a server-side database, which maintains ticket and event availability information and processes ticket purchases. The Java Servlet handles form submission and updates the database with seat purchases.

The **HTML, JS, DOM** demo shows how SVG manages and displays data, generating SVG code from data on the fly. Although this kind of application can be written in a variety of different ways, SVG provides client-side processing to maintain and display the data, reducing the load on the server as well as overall latency. Using the DOM, developers can build documents, navigate their structure, and add, modify, or delete elements and content.

The **PHP, MySQL** demo shows the use of database driven SVG generation utilizing MySQL. It randomly generates a map of a European country. Each time you reload the page you will see a different country.

**Exercises**
Download and install the Adobe SVG Viewer. Once the Adobe SVG Viewer has been installed, go to this page to test that the install was successful:
http://www.adobe.com/svg/viewer/install/svgtest.html

If your primary browser is Internet Explorer, you must download the 6.0 version at
http://download.adobe.com/pub/adobe/magic/svgviewer/win/6.x/6.0x38363/en/SVGView.exe

After it has been installed you must copy the NPSVG6.dll and NPSVG6.zip files to your browser's plug-ins folder. These files are normally located in C:\Program Files\Common Files\Adobe\SVG Viewer 6.0\Plugins\.

2. Create your own standalone SVG file to produce an image containing a circle within a rectangle.
3. Create your own stand-alone SVG file. Use 3 circles and 1 path element to create a yellow smiley face with black eyes and a black mouth. Use a text element so that the message “Have a nice day!” appears below the smiley face.
   - Hint: Because <path> elements can be difficult to write, here is a sample path you can utilize:
     - <path d="M 100, 120 C 100,120 140, 140 180,120" style="fill:none;stroke:black;stroke-width:1"/>

References
1. libsrsvg - free, open source SVG rendering library (http://libsrsvg.sourceforge.net/)
2. Barry Kauler blog (http://bkhome.org/blog/?viewDetailed=02088)
3. 2D Apollonian gasket with four identical circles by Guillaume JACQUENOT 1 (http://www.mathworks.com/matlabcentral/fileexchange/15958-2d-apollonian-gasket-with-four-identical-circles)
4. W3C SVG 1.1 Recommendation: SVG Namespace, Public Identifier and System Identifier (https://www.w3.org/TR/SVG/single-page.html#intro-NamespaceAndDTDIdentifiers), see also W3C SVG 2 Editor’s Draft: SVG namespace and DTD (https://svgwg.org/svg2-draft/single-page.html#intro-Namespace) (06 February 2013)
5. SVG authoring and web server configuration guidelines. Jonathan Watr Don’t include a DOCTYPE declaration (https://jwat.org/svg/authoring/#doctype-declaration)
7. validator.w3.org (http://validator.w3.org/)
   - Collection of free SVG vectors http://www.easyvectors.com/
   - The Adobe SVG Zone http://www.adobe.com/svg/
   - Cartographers on the Net http://www.carto.net/
   - Learn SVG http://www.learnsvg.com/
   - SVG authoring and web server configuration guidelines http://jwat.org/svg/authoring/
   - W3C SVG Tutorial http://www.w3schools.com/svg/default.asp
   - W3C Synchronized Multimedia http://www.w3.org/AudioVideo/
   - Mozilla SVG Project http://www.mozilla.org/projects/svg/
   - svg.startpagina.nl http://svg.startpagina.nl/

Voicexml examples

According to the W3C, "VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications."

Here are two short examples of VoiceXML. The first is the always fun example, "Hello World":

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
  <block>Hello world</block>
</form>
```
The top-level element is `<vxml>`, which is mainly a container for dialogs. The two main types of dialogs are forms and menus. Forms present information and gather input. Menus offer choices of what to do next. This example has a single form, which contains a block that synthesizes and presents "Hello World!" to the user. Since the form does not specify a dialog after "Hello World", the conversation ends.

Our second example asks the user for a choice of drink and then submits it to a server script:

Form example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <form>
    <field name="drink">
      <prompt>Would you like coffee, tea, milk, or nothing?</prompt>
      <grammar type="application/x-gsl" mode="voice">
        <![CDATA[
          [coffee] {<drink "Coffee">} 
          [tea] {<drink "Tea">} 
          [milk] {<drink "Milk">} 
          [nothing] {<drink "Nothing">}
        ]]>"
      </grammar>
    </field>
    <block>
      <submit next="http://www.drink.example.com/drink2.asp"/>
    </block>
  </form>
</vxml>
```

A field is an input field. The user must provide a value for the field before the next element in the form is referenced or executed. Here is an example of a simple interaction:

- C (computer): Would you like coffee, tea, milk, or nothing?
- H (human): Orange juice.
- C: I did not understand what you said. (a platform-specific default message.)
- H: Tea
- C: (continues in document drink2.asp)

Menu example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <menu>
    <property name="inputmodes" value="dtmf"/>
    <prompt>
      For sports press 1, for weather press 2, for Stargazer astrophysics press 3.
    </prompt>
    <choice dtmf="1" next="http://www.sports.example.com/vxml/start.vxml"/>
    <choice dtmf="2" next="http://www.weather.example.com/intro.vxml"/>
    <choice dtmf="3" next="http://www.stargazer.example.com/astronews.vxml"/>
  </menu>
</vxml>
```

The computer, or receiver, recognizes the number and sends a message to trigger the next dialog, according to which number was chosen. Here is what a typical conversation would look like:

- H: 4
- C: I did not understand what value you typed. (a platform-specific default message.)
- H: 1 "sports"
- C: (proceeds to http://www.sports.example.com/vxml/start.vxml)

The beginning of VoiceXML

VoiceXML began in 1995 as an XML-based dialog design language. It was mainly used to simplify the speech recognition applications in an AT&T project called Phone Markup Language (PML). After the creation of this language, some other companies worked on their own PML-like languages such as Lucent, Motorola (VoxML), IBM (SpeechML), HP (TalkML) and PipeBeach (VoiceHTML). Since 1998, The VoiceXML Forum has been developed by AT&T, IBM, Lucent, and Motorola to define a standard dialog design language that developers could use to build conversational applications. They chose XML as the basis for this effort because it was clear to them that this was the direction technology was going. By 2000, the VoiceXML Forum released VoiceXML 1.0 to the public and submitted it to the W3C to set the language as an international standard. This implementation allowed the release of VoiceXML 2.0, based on input from W3C member companies, W3C working groups, and all kinds of developers.

Introduction
VoiceXML is created to generate audio dialogs that allows the use of synthesized speech, digitized audio, recognition of spoken and DTMF (Dual Tone Multi-Frequency Touch-tone or push-button dialing.) In Layman's Terms, VoiceXML allows the use of computer speech, recorded audio, human speech, and telephones as input and output devices. Pushing a button on a telephone keypad generates a sound that is a combination of two tones, one high frequency and the other low frequency) key input, recording of spoken input, telephony, and mixed initiative conversations.

**VoiceXML architectural model**

The architectural model assumed by this document has the following components:

![VoiceXML Architectural Model Diagram](image)

A document server (e.g. a Web server) processes requests from a client application, the VoiceXML Interpreter, through the VoiceXML interpreter context. The server produces VoiceXML documents in reply, which are processed by the VoiceXML interpreter. The VoiceXML interpreter context may monitor user inputs in parallel with the VoiceXML interpreter. For example, one VoiceXML interpreter context may always listen for a special escape phrase that takes the user to a high-level personal assistant, and another may listen for escape phrases that alter user preferences like volume or text-to-speech characteristics.

The implementation platform is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter. For instance, in an interactive voice response application, the VoiceXML interpreter context may be responsible for detecting an incoming call, acquiring the initial VoiceXML document, and answering the call, while the VoiceXML interpreter conducts the dialog after answer. The implementation platform generates events in response to user actions (e.g. spoken or character input received, disconnect) and system events (e.g. timer expiration). Some of these events are acted upon by the VoiceXML interpreter itself, as specified by the VoiceXML document, while others are acted upon by the VoiceXML interpreter context.

**The Goals of VoiceXML**

VoiceXML's main goal is to bring the full power of Web development and content delivery to voice response applications, and to free the authors of such applications from low-level programming and resource management. VoiceXML sets an integration environment between voice services and data services taking advantage of the client-server paradigm. A voice service can be defined as a sequence of interactive dialogs between a user and an implementation platform. The dialogs are stored in document servers, allowing an independent structure from the implementation platform. These servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document interacts with the dialogs from the server using a VoiceXML interpreter. The inputs from the user generates requests to the document server, and finally, the document server replies with another VoiceXML document to continue the user’s session with other dialogs.

VoiceXML is a markup language that:

- Minimizes client/server interactions generating all kinds of interactions per document.
- Shields application authors from low-level, and platform-specific details.
- Separates user interaction code (in VoiceXML) from service logic (e.g. CGI scripts).
- Allows multiparty development, becoming a common language for content providers, tool providers, and platform providers.
- Offers ease of use for simple interactions, and yet provides language features to support complex dialogs.

While VoiceXML strives to accommodate the requirements of a majority of voice response services, services with stringent requirements may best be served by dedicated applications that employ a finer level of control.

**Principles of Design**

VoiceXML is an XML application [XML]. These are some of the capabilities, or abilities VoiceXML carries:

- The language promotes portability of services through abstraction of platform resources.
- The language accommodates platform diversity in supported audio file formats, speech grammar formats, and URI schemes.
- The language makes it easy to create common types of interactions.
- The language has well-defined standards of wording and syntax that allows for the author's intent regarding the behavior of interactions with the user much easier.
- The language recognizes semantic interpretations from all types of grammars and makes this information available to the application.
- The language has a control flow mechanism.
- The language enables a separation of service logic from interaction behavior.
- It is not intended for intensive computation, database operations, or legacy system operations. These are assumed to be handled by resources outside the document interpreter, e.g. a document server.
- General service logic, state management, dialog generation, and dialog sequencing are assumed to reside outside the document interpreter.
- The language provides ways to link documents and submit data to server scripts using URIs.
- VoiceXML provides ways to identify exactly which data to submit to the server, and which HTTP method (GET or POST) to use in the submittal.
- The language does not require document authors to explicitly allocate and deallocate dialog resources.

**Implementation Platform Requirements**

This section outlines the hardware/software requirements to support a VoiceXML interpreter:
The purpose was to acquire documents from the VoiceXML interpreter, requiring the support of the 'http' URI protocol. There will be some cases in which the document request is generated by the interpretation of a VoiceXML document, but it can also be generated in response to events outside the scope of the language, like an incoming phone call. When issuing document requests via http, the interpreter context identifies itself using the "User-Agent" header variable with the value "<name>/<version>", for example, "acme-browser/1.2."

Audio output: An implementation platform must support audio output using audio files and text-to-speech (TTS). The platform must be able to freely sequence TTS and audio output. If an audio output resource is not available, an error.noresource event must be thrown. These files are referenced by a particular URI.

Audio input: An implementation platform needs to find the way to detect and report character and/or spoken input simultaneously. It also needs to control input detection interval duration with a timer whose length is specified by a VoiceXML document.

- Platforms must support the XML form of DTMF grammars described in the W3C Speech Recognition Grammar Specification SRGS.
- It must be able to receive speech recognition grammar data dynamically.
- It can support other formats such as the ISpeech Grammar Format or proprietary formats.
- It must be able to record audio received from the user.
- The platform should be able to support making a third party connection through a communications network, such as the telephone.

Transfer: The platform should be able to support making a third party connection through a communications network, such as the telephone.

Concepts

A VoiceXML document is a conversational finite state machine, in which the user is always in one conversational state, or dialog, at a time. Each dialog determines the next dialog to transition to. Transitions can be defined using URIs, which define the next document and dialog to use. When there are no more dialogs, or there is an element that explicitly exits the conversation, the execution is terminated. A VoiceXML document is primarily composed of top-level elements called dialogs.

There are two types of dialogs: forms and menus. A document may also have:

- <meta> elements.
- <metadata> elements.
- variable elements.
- <script> elements.
- <property> elements.
- <catch> elements.
- <link> elements.

Forms define an interaction that collects values from a set of field item variables. Each field may specify a grammar that defines the allowable inputs for that field.

Menus display the information to the user with a choice of options and then transitions to another dialog based on the selected choice. Each dialog has involved a series of speech and/or DTMF grammars, which are active only when the user is in that dialog.

A subdialog is like a function call because it provides a way to creating and invoking a new interaction, and returning to the original dialog. Variable instances, grammars, and state information are saved and are available upon returning to the calling document. Subdialog can be used to create a confirmation sequence that may require a database query, create a set of components that may be shared among documents in a single application, or possibly to create a reusable library of dialogs shared among many applications.

A session begins when the user starts to interact with a VoiceXML interpreter context, continues as documents are loaded and processed, and ends when requested by the user, a document, or the interpreter context.

An application is a set of documents sharing the same application root document. Whenever the user interacts with a document in an application, its application root document is also loaded. The application root document remains loaded while the user is transitioning between other documents in the same application, and it is unloaded when the user transitions to a document that is not in the application.

Grammars: Each dialog has one or more speech and/or DTMF grammars associated with it. In machine directed applications, each dialog's grammars are active only when the user is in that dialog. In mixed initiative applications, where the user and the machine alternate in determining what to do next, some of the dialogs are flagged to make their grammars active (i.e., listened for) even when the user is in another dialog in the same document, or on another loaded document in the same application. In this situation, if the user says something matching another dialog's active grammars, execution transitions to that other dialog, with the user's utterance treated as if it were said in that dialog. Mixed initiative adds flexibility and power to voice applications.

Events: VoiceXML allows the user to fill forms in the traditional way of user input and defines mechanisms for handling events not covered by the form mechanism. Events can be thrown when the user does not respond, does not respond correctly, or requests assistance. Similarly, the VoiceXML interpreter also can throw events if it finds a semantic error in a VoiceXML document using catch elements that allow the interpreter to trigger such events.

A link specifies a grammar that is active whenever the user interacts with it. If user input matches the link's grammar, control transfers to the link's destination URI. A link can be used to throw an event or go to a destination URI.

VoiceXML elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;assign&gt;</td>
<td>Assign a variable a value</td>
</tr>
<tr>
<td>&lt;audio&gt;</td>
<td>Play an audio clip within a prompt</td>
</tr>
<tr>
<td>&lt;block&gt;</td>
<td>A container of (non-interactive) executable code</td>
</tr>
<tr>
<td>&lt;catch&gt;</td>
<td>Catch an event</td>
</tr>
</tbody>
</table>

For more information about the elements go to W3C page.

http://www.w3.org/TR/2004/REC-voicexml20-20040316/
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;clear&gt;</td>
<td>Clear one or more form item variables</td>
</tr>
<tr>
<td>&lt;disconnect&gt;</td>
<td>Disconnect a session</td>
</tr>
<tr>
<td>&lt;else&gt;</td>
<td>Used in &lt;if&gt; elements</td>
</tr>
<tr>
<td>&lt;elseif&gt;</td>
<td>Used in &lt;if&gt; elements</td>
</tr>
<tr>
<td>&lt;enumerate&gt;</td>
<td>Shorthand for enumerating the choices in a menu</td>
</tr>
<tr>
<td>&lt;error&gt;</td>
<td>Catch an error event</td>
</tr>
<tr>
<td>&lt;exit&gt;</td>
<td>Exit a session</td>
</tr>
<tr>
<td>&lt;field&gt;</td>
<td>Declares an input field in a form</td>
</tr>
<tr>
<td>&lt;filled&gt;</td>
<td>An action executed when fields are filled</td>
</tr>
<tr>
<td>&lt;form&gt;</td>
<td>A dialog for presenting information and collecting data</td>
</tr>
<tr>
<td>&lt;goto&gt;</td>
<td>Go to another dialog in the same or different document</td>
</tr>
<tr>
<td>&lt;grammar&gt;</td>
<td>Specify a speech recognition or DTMF grammar</td>
</tr>
<tr>
<td>&lt;help&gt;</td>
<td>Catch a help event</td>
</tr>
<tr>
<td>&lt;if&gt;</td>
<td>Simple conditional logic</td>
</tr>
<tr>
<td>&lt;initial&gt;</td>
<td>Declares initial logic upon entry into a (mixed initiative) form</td>
</tr>
<tr>
<td>&lt;link&gt;</td>
<td>Specify a transition common to all dialogs in the link’s scope</td>
</tr>
<tr>
<td>&lt;log&gt;</td>
<td>Generate a debug message</td>
</tr>
<tr>
<td>&lt;menu&gt;</td>
<td>A dialog for choosing amongst alternative destinations</td>
</tr>
<tr>
<td>&lt;meta&gt;</td>
<td>Define a metadata item as a name/value pair</td>
</tr>
<tr>
<td>&lt;metadata&gt;</td>
<td>Define metadata information using a metadata schema</td>
</tr>
<tr>
<td>&lt;noinput&gt;</td>
<td>Catch a noinput event</td>
</tr>
<tr>
<td>&lt;nomatch&gt;</td>
<td>Catch a nomatch event</td>
</tr>
<tr>
<td>&lt;object&gt;</td>
<td>Interact with a custom extension</td>
</tr>
<tr>
<td>&lt;option&gt;</td>
<td>Specify an option in a &lt;field&gt;</td>
</tr>
<tr>
<td>&lt;param&gt;</td>
<td>Parameter in &lt;object&gt; or &lt;subdialog&gt;</td>
</tr>
<tr>
<td>&lt;prompt&gt;</td>
<td>Queue speech synthesis and audio output to the user</td>
</tr>
<tr>
<td>&lt;property&gt;</td>
<td>Control implementation platform settings.</td>
</tr>
<tr>
<td>&lt;record&gt;</td>
<td>Record an audio sample</td>
</tr>
<tr>
<td>&lt;reprompt&gt;</td>
<td>Play a field prompt when a field is re-visited after an event</td>
</tr>
<tr>
<td>&lt;return&gt;</td>
<td>Return from a subdialog.</td>
</tr>
<tr>
<td>&lt;script&gt;</td>
<td>Specify a block of ECMAScript client-side scripting logic</td>
</tr>
<tr>
<td>&lt;subdialog&gt;</td>
<td>Invoke another dialog as a subdialog of the current one</td>
</tr>
<tr>
<td>&lt;submit&gt;</td>
<td>Submit values to a document server</td>
</tr>
<tr>
<td>&lt;throw&gt;</td>
<td>Throw an event.</td>
</tr>
<tr>
<td>&lt;transfer&gt;</td>
<td>Transfer the caller to another destination</td>
</tr>
<tr>
<td>&lt;value&gt;</td>
<td>Insert the value of an expression in a prompt</td>
</tr>
<tr>
<td>&lt;variable&gt;</td>
<td>Declare a variable</td>
</tr>
<tr>
<td>&lt;vxml&gt;</td>
<td>Top-level element in each VoiceXML document</td>
</tr>
</tbody>
</table>

**One Document Execution**

Document execution starts with the first dialog by default. As each dialog executes, the next dialog is determined. When a dialog doesn’t reference another dialog, document execution stops.

Here is the "Hello World!" example expanded to illustrate VoiceXML execution. It now has a document level variable called "hi" which holds the greeting. Its value is used as the prompt in the first form. Once the first form plays the greeting, it goes to the form named "say_goodbye", which prompts the user with "Goodbye!" Because the second form does not have a transition to another dialog, the document execution ceases.
Variables and Expressions

VoiceXML variables are in all respects equivalent to ECMAScript variables: they are part of the same variable space. VoiceXML variables can be used in a `<script>` just as variables defined in a `<script>` can be used in VoiceXML. Declaring a variable using `var` is equivalent to using a `var` statement in a `<script>` element. `<script>` can also appear everywhere that `var` can appear. VoiceXML variables are also declared by form items.

The variable naming convention is as in ECMAScript, but names beginning with the underscore character ("_") and names ending with a dollar sign ("$") are reserved for internal use. VoiceXML variables, including form item variables, must not contain ECMAScript reserved words. They must also follow ECMAScript rules for referential correctness. For example, variable names must be unique and their declaration must not include a dot - "var x.y" is an illegal declaration in ECMAScript. Variable names which violate naming conventions or ECMAScript rules cause an 'error.semantic' event to be thrown.

Variables are expressed using the `var` element:

```xml
<var
    name='"room_number"'/>
<var
    name='"avg_mult"'
    expr='2.2'/>
<var
    name='"state"
    expr='"Georgia"'/>
```

Attributes of `<vxml>` include:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of VoiceXML of this document (required). The current version number is 2.0.</td>
</tr>
<tr>
<td>base</td>
<td>Defines a base URI, to be used when resolving relative URIs in the document.</td>
</tr>
<tr>
<td>xmlns</td>
<td>The designated namespace for VoiceXML (required). The namespace for VoiceXML is defined to be <a href="http://www.w3.org/2001/vxml">http://www.w3.org/2001/vxml</a>.</td>
</tr>
<tr>
<td>xmlns:xsi</td>
<td>Used along with xsi:schemaLocation to indicate the location of the schema for the VoiceXML namespace.</td>
</tr>
<tr>
<td>xsi:schemaLocation</td>
<td>Used along with xmlns:xsi to indicate the location of the schema for the VoiceXML namespace.</td>
</tr>
<tr>
<td>application</td>
<td>The URI of this document’s application root document, if any.</td>
</tr>
</tbody>
</table>

`<field>` Element

A field specifies an input item to be gathered from the user. Some attributes of this element are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The form item variable in the dialog scope that will hold the result. The name must be unique among form items in the form.</td>
</tr>
<tr>
<td>expr</td>
<td>The initial value of the form item variable, default is ECMAScript undefined.</td>
</tr>
<tr>
<td>cond</td>
<td>An expression that must evaluate to true after conversion to boolean in order for the form item to be visited. The form item can also be visited if the attribute is not specified.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of field, i.e., the name of a built in grammar type</td>
</tr>
</tbody>
</table>

`<grammar>` Element

The `<grammar>` element is used to provide a speech grammar that

- specifies the expressions that a user may use to perform an action or supply information
- returns a corresponding semantic interpretation such as simple values (strings), attribute-value pairs (day, month, and year), or nested objects.

Some attributes of the `<grammar>` element are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Defines the version of the grammar.</td>
</tr>
<tr>
<td>xml:lang</td>
<td>The identifier for the language of that specific grammar (&quot;fr-CA&quot; for Canadian French.)</td>
</tr>
<tr>
<td>mode</td>
<td>Defines the mode of the grammar following the modes of the W3C Speech Recognition Grammar Specification SRGS.</td>
</tr>
<tr>
<td>root</td>
<td>Defines the root rule of the grammar.</td>
</tr>
<tr>
<td>tag-format</td>
<td>Defines the tag content format for all tags within the grammar.</td>
</tr>
<tr>
<td>xml:base</td>
<td>Declares the base URI from which relative URIs in the grammar are resolved. This base declaration has precedence over the <code>&lt;vxml&gt;</code> base URI declaration.</td>
</tr>
</tbody>
</table>
<block> Element

This element is a form item. It contains executable content that is executed if the block’s form item variable is undefined and the block’s cond attribute, if any, evaluates to true.

Welcome to Flamingo, your source for lawn ornaments.

The form item variable is automatically set to true just before the block is entered. Therefore, blocks are typically executed when the form is called. Sometimes you may need more control over blocks. To do this, you can name the form item variable, and set or clear it to control execution of the <block>. This variable is declared in the dialog scope of the form. Attributes of <block> include:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the form item variable used to track whether this block is eligible to be executed; defaults to an inaccessible internal variable.</td>
</tr>
<tr>
<td>expr</td>
<td>The initial value of the form item variable; default is ECMAScript undefined.</td>
</tr>
<tr>
<td>cond</td>
<td>An expression that must evaluate to true after conversion to boolean in order for the form item to be visited.</td>
</tr>
</tbody>
</table>

<prompt> Element

This element controls the output of synthesized speech and prerecorded audio. Prompts are queued for play, and interpretation will start when the user provides an input. Here is an example of a prompt:

*please say your name.*

You can leave out the <prompt> ... </prompt> if:

- There is no need to specify a prompt attribute (like bargein), and
- The prompt consists entirely of PCDATA (contains no speech markups) or consists of just an <audio> or <value> element.

For instance, these are also prompts:

*please say your name.*

But sometimes you have to use the <prompt> tags when adding embedded speech markups, such as:

*please <emphasis>say</emphasis> your city.*

The <prompt> element has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Expression that must evaluate to true after conversion to boolean in order for the prompt to be played. Default is true.</td>
</tr>
<tr>
<td>count</td>
<td>Number that allows you to emit different prompts if the user is doing something repeatedly. If omitted, it defaults to &quot;1&quot;.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout that will be used for the following user input. The default noinput timeout is platform specific.</td>
</tr>
<tr>
<td>xml:lang</td>
<td>The language for the prompt identifier.</td>
</tr>
<tr>
<td>xml:base</td>
<td>Declares the base URI from which relative URIs in the prompt are resolved.</td>
</tr>
</tbody>
</table>

Exercises

1. Create a VoiceXML document in which you give the user three different options to choose from the keyboard. The user must choose one option between hotels, museums or restaurants. Use forms for this exercise. Hint: this exercise needs to use the option element tag Example: <option dtmf="1" value="varName"> Display name </option>

2. Create a VoiceXML document in which you give the user three different options to choose from the keyboard. The user must choose one option between hotels, museums or restaurants. Use menu dialogs for this exercise.

References

- VoiceXML 2.0 Recommendation: http://www.w3.org/TR/2004/REC-voicexml20-20040316/
- W3C Voice Browser WG: http://www.w3.org/Voice/
- VoiceXML Forum: http://www.voicexml.org
- Tellme studio: http://studio.tellme.com/
- BeVocal cafe: http://cafe.bevocal.com/
- VoiceXML Italian User Group: http://www.vxmlitalia.com

XML - Managing Data Exchange

<table>
<thead>
<tr>
<th>Related Topics</th>
<th>Get Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapters</td>
<td>Computer Science Home</td>
</tr>
<tr>
<td></td>
<td>To do list</td>
</tr>
</tbody>
</table>
Learning objectives

Upon completion of this chapter, you will be able to

- Learn the basics of DocBook
- Create a DocBook document by using the DocBook XML DTD
- Convert a text document to a DocBook document
- Use XSL Stylesheets to transform a DocBook XML document to multiple formats as HTML, PDF or presentation slides.

Introduction

DocBook is a general purpose XML and SGML vocabulary particularly well suited to books, articles, and papers. It has a large, powerful and easy to understand Document Type Definition (DTD), and its main structures correspond to the general concept of what constitutes a book. DocBook is a substantial subject that we can't exactly cover in a few pages. Thus, for the purposes of this chapter, we will talk about creating a simple DocBook document with major elements in the Docbook DTD and the details of publishing the document in order to give you a feel about DocBook. If you would like to study the subject further, we suggest you to have a look at the references provided at the end of the chapter.

What is DocBook?

- DocBook enables you to author and store document content in a presentation-neutral form that captures the logical structure of the content.
- It has an easy-to-understand and widely used DTD. The DocBook tags are applied so that they have a certain "common sense" semantic content, at least to English speakers.
- There are no official versions of the DocBook W3C XML Schema at this time. The DocBook Technical Committee is planning to offer an official Schema in the DocBook V5.0 time frame. The examples provided in this chapter will use the current official DTD.

DTD vs. Schema:

A DTD is the XML Document Type Definition contains or points to markup declarations that provide a grammar for a class of documents. A Schema is a set of shared vocabularies that allow machines to carry out rules made by people. It provides a means for defining the structure, content and semantics of XML documents. In summary, schemas are a richer and more powerful means of describing information than DTDs.

Table 1: Here is a simple xml document

```
<author>
  <firstname>Rusen</firstname>
  <lastname>Gul</lastname>
</author>
```

Table 2: Here is the DTD for this document

```
<!ELEMENT author(firstname, lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname(#PCDATA)>
```

Table 3: And here is the SCHEMA

```
<xsd:element name="author">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Output formats for DocBook

XSL (Extensible Style Language) stylesheets can transform DocBook XML into the following formats:

- HTML
- HTML Help (for Windows Help)
- Java Help
- XHTML
DSSSL (Document Style Semantics and Specification Language) stylesheets can transform DocBook SGML into the following formats:

- HTML
- MIF
- RTF
- TeX

A Brief History

DocBook was created around 1991 by HaL Computer Systems and O'Reilly & Associates. It was developed primarily for the purpose of holding the results of troff conversion of UNIX documentation, so that the files could be interchanged. Now it is maintained by OASIS. The official web site for DocBook is http://www.oasis-open.org/docbook/

DocBook is used for:

1. Books for print/trade publication. Many authors are using DocBook for writing books of all kinds, in various print and online formats, worldwide. Some examples include:
   - Processing XML with Java (http://www.ibiblio.org/xmlbooks/xmljava/chapters/)
   - DocBook: The Definitive Guide (http://www.docbook.org/)
   - Dive into Python (http://www.diveintopython.org/)
2. Articles, theses and dissertations
3. Maintaining websites
4. Producing presentation slides, printed handouts, and whitepapers
5. Documentation for commercial software and hardware

DocBook Tools

DocBook is officially available as a DTD for both XML and SGML. You can download both the latest DocBook XML DTD and DocBook SGML DTD from the official DocBook site at OASIS. The examples provided in this chapter will use DocBook XML DTD. Some experimental DocBook schemas are available at sourceforge.net. DocBook is supported by a number of commercial and open source tools. Easily customizable and extensible "standard" DocBook Stylesheets are available from the DocBookOpenRepository along with the other free open source tools. See DocBookTools (http://wiki.docbook.org/topic/DocBookTools) on the DocBook Wiki for a more complete list of commercial and open source tools.

Other Free Tools:

1. XSLTProc: One of the most known and fast processor. Available at http://xmlsoft.org/XSLT/.

Commercial Tools:

2. RenderX XEP: FO to PDF rendering engine. Available at RenderX website (http://www.renderx.com/products.html).

SGML vs XML

The syntax of SGML and DTD is very similar but not identical. The biggest difference between the DocBook DTD for SGML and the one for XML is that the SGML DTD contains SGML exclusions in some content models.

Example: SGML DTD excludes `<footnote>` as a descendent of `<footnote>`, because it doesn't make much practical sense to have footnotes within footnotes. XML DTDs can't contain exclusions, so if you're authoring using the DocBook XML DTD, it's possible to produce documents containing some valid-but-not-logical markup like footnotes within footnotes.

Creating a DocBook Document

In order to get started, you will need:

- The DocBook XML DTD. Although it's optional to use one, DTD's are useful when one wants to validate a document to check that it conforms to the DTD to which one claims it conforms. Hence, the DocBook DTD can be used to validate that a purported DocBook document. DocBook XML 4.2 is the current version of DocBook DTD. Download at the official DocBook website (http://www.oasis-open.org/docbook/xml/)
- The DocBook XSL stylesheets are maintained primarily by Norman Walsh. There are two sets of stylesheets: XSL and DSSSL. Download the latest version XSL 1.65.1 at Sourceforge.net (http://sourceforge.net/project/showfiles.php?group_id=21935)
- An XSLT processor (covered in the further sections)

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis-open.org/docbook/xmll/4.2/docbookx.dtd">
<book>
  <bookinfo>
```

Table 4: A simple DocBook Book, "book.xml"
Let's examine the details of a typical DocBook document. Standard header to a DocBook XML file is a DocType declaration:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "‐//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis‐open.org/docbook/xml/4.2/docbookx.dtd">
```

Let’s describe the different parts of the file. The Owner, Keywords, and Language are defined using special documentation.

Owner: Oasis
Keyword Description: DTD DocBook XML V4.2
Language: EN - English

Caution! If you are not online, you need to change the URL system identifier to path where DTD is installed:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "‐//OASIS//DTD DocBook XML V4.2//EN"
/usr/share/sgml/docbook/xml‐dtd‐4.2/docbookx.dtd">
```

## Breaking a Document into Physical Portions

Before getting started, here is a useful tip! For the purposes of convenience and performance, you might consider breaking a document into physical chunks and work on each chunk separately. If you have a book that consists of three chapters and two appendices, you might create a file called `book.xml`, which looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "‐//OASIS//DTD DocBook V4.2//EN"
[<!ENTITY chap1 SYSTEM "chap1.xml">
<!ENTITY chap2 SYSTEM "chap2.xml">
<!ENTITY appa SYSTEM "appa.xml">
<!ENTITY appb SYSTEM "appb.xml">
]>
<book>
  <title>A Physically Divided Book</title>
  &chap1;
  &chap2;
  &appa;
  &appb;
</book>
```

Table 5: A simple DocBook article, “article.xml”

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "‐//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis‐open.org/docbook/xml/4.2/docbookx.dtd">
<article>
  <articleinfo>
    <title>A Simple Approach To DocBook</title>
    <author>
      <firstname>Rusen</firstname>
      <surname>Gul</surname>
    </author>
  </articleinfo>
  <para>This is the introductory paragraph of my article.</para>
  <sect1>
    <title>First Section</title>
    <para>This is a paragraph in the first section.</para>
    <sect2>
      <title>This is the title for section2.</title>
      <para>This is a paragraph in section2.</para>
    </sect2>
    <sect2>...
    </sect2>
  </sect1>
  <sect1>...
  </sect1>
</article>
```

Table 6: A physically divided book, “dividedbook.xml”

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "‐//OASIS//DTD DocBook V4.2//EN"
[<!ENTITY chap1 SYSTEM "chap1.xml">
<!ENTITY chap2 SYSTEM "chap2.xml">
<!ENTITY appa SYSTEM "appa.xml">
<!ENTITY appb SYSTEM "appb.xml">
]>
<book>
  <title>A Physically Divided Book</title>
  &chap1;
  &chap2;
  &appa;
  &appb;
</book>
```
You can then write the chapters and appendices conveniently in separate files. This is why DocBook is well suited to large contents. Note that these separate files do not and must not have document type declarations.

For example, Chapter 1 might begin like this:

```xml
.chapter id="ch1">
  .title My First Chapter</title>
  .para My first paragraph.</para>...
</chapter>```

### Breaking a Document into Logical Portions


There are—literally—hundreds of DocBook elements. This is what makes docBook very powerful. We will try to cover the major ones here and let you review the rest on your own. Firstly, a classification; DocBook Elements can be divided broadly into these categories:

<table>
<thead>
<tr>
<th>Sets</th>
<th>Books</th>
<th>Divisions</th>
<th>Components</th>
<th>Sections</th>
<th>Meta-information Elements</th>
<th>Block Elements</th>
<th>Inline Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection of books</td>
<td>books</td>
<td>divide books into parts</td>
<td>divide books or divisions into chapters</td>
<td>subdivide components</td>
<td>contain information about other elements</td>
<td>occur at paragraph level</td>
<td>used to mark up running text</td>
</tr>
</tbody>
</table>

### Major DocBook Elements

**Set: A collection of books**

`Set` is the very top of the DocBook structural hierarchy. There’s nothing that contains a `Set`.

**Some children elements:** Book, SetIndex, SetInfo, Subtitle, Title, TitleAbbrev, ToC (table of contents).


Table 7: `<set>` element, "lordoftherings.xml"

```xml
<!DOCTYPE set PUBLIC "‐//OASIS//DTD DocBook V4.2//EN">
<set>
  .title Lord of the Rings</title>
  .setinfo
    .author J.R. Tolkien</author>
  </setinfo>
  .book</title>The Fellowship of the Ring</title> ... </book>
  .book</title>The Two Towers</title> ... </book>
  .book</title>Return of the King</title> ... </book>
</set>
```

**Book: A book**

A `Book` is probably the most common top-level element in a document. The DocBook definition of a book is very loose and general. It gives you free rein by not imposing a strict ordering of elements.

**Some children elements:** Appendix, Article, Bibliography, BookInfo, Chapter, Colophon, Dedication, Glossary, Index, LoT, Part, Preface, Reference, SetIndex, Subtitle, Title, TitleAbbrev, ToC.


```xml
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "‐//OASIS//DTD DocBook V4.2//EN">
<book>
  .title XML – Managing Data Exchange</title>
  .titleabbrev XML</titleabbrev>
  .bookinfo
    .legalnotices><para>No notice is required.</para></legalnotices>
    .author Rusen, Rusen</author>
  </bookinfo>
  .dedication
    .para This book is dedicated to MIST 7700 class of 2004 at UGA.</para>
  </dedication>
  .preface
    .title Forward</title>
    .para The book aims to fulfill the need for an introductory XML textbook. It contains the basics of XML as well as several tools using XML.</para>
  </preface>
  .chapter
    .title Introduction</title>
    .para At least one chapter, reference, part, or article is required.</para>
</chapter>"
Division: A collection of parts and references (optional)

Divisions are the first hierarchical level below Book.

Children elements: Part (contain components), Reference (contain RefEntrys)

Components: Chapter-like elements of a Book or Part

These are Preface, Chapter, Appendix, Glossary, Bibliography, and Article. Components generally contain block elements -or sections, and some can contain navigational components and RefEntrys.

Table 9: <Bibliography> element, "references.xml"

```xml
<!DOCTYPE bibliography PUBLIC "-//OASIS//DTD DocBook 4.2//EN">
<bibliography>
  <title>References</title>
  <bibliomixed>
    <bibliomset relation=article>
      <surname>Watson</surname>
      <firstname>Richard</firstname>
      <title role=article>Managing Global Communities</title>
    </bibliomset>
    <bibliomset relation=journal>
      <title>The World Wide Web Journal</title>
      <volumenum>2</volumenum>
      <issuenum>1</issuenum>.
      <publishername>O'Reilly & Associates, Inc.</publishername>
      <pubdate>Winter, 1996</pubdate>
    </bibliomset>
  </bibliomixed>
</bibliography>
```

Sections: Several sectioning elements

a. Sect1…Sect5 elements - the most common sectioning elements that can occur in most component-level elements. These numbered section elements must be properly nested (Sect2s can only occur inside Sect1s, Sect3s can only occur inside Sect2s, and so on).

b. Section element - an alternative to numbered sections Sections are recursive, meaning that you can nest them to any depth desired.

c. SimpleSect element - a terminal section that can occur at any level SimpleSect cannot have any other sectioning element nested within it.

d. BridgeHead element - a section title without any containing section

e. RefSect1…RefSect3 elements - numbered section elements in RefEntrys f. GlossDiv, BiblioDiv, and IndexDiv elements - do not nest

Please see Table 4 and Table 5 for examples.


Meta-Information Elements – contain bibliographic information

All of the elements at the section level and above include a wrapper for meta-information about the content. Examples of meta-wrappers: BookInfo, ArticleInfo, ChapterInfo, PrefaceInfo, SetInfo, GlossaryInfo.

Table 10: <bookinfo> element

```xml
<!DOCTYPE bookinfo PUBLIC "-//OASIS//DTD DocBook V4.2//EN">
<bookinfo>
  <title>XML – Managing Data Exchange</title>
  <authorgroup>
    <author>
      <firstname>Richard</firstname>
      <surname>Watson</surname>
    </author>
    <author>
      <firstname>Hendrik</firstname>
      <surname>Fischer</surname>
    </author>
    <author>
      <firstname>Rusen</firstname>
      <surname>Gul</surname>
    </author>
  </authorgroup>
  <edition>Introduction to XML - Version 1.0</edition>
  <pubdate>1999</pubdate>
  <copyright>1999</copyright>
  <year>2000</year>
  <year>2001</year>
  <year>2002</year>
  <year>2003</year>
  <holder>O'Reilly & Associates, Inc.</holder>
</bookinfo>
```
Block vs. Inline Elements

There are two classes of paragraph-level elements: block and inline.

Block elements are usually presented with a paragraph break before and after them. Most can contain other block elements, and many can contain character data and inline elements. Examples of block elements are: Paragraphs, lists, sidebars, tables, and block quotations.

Inline elements are generally represented without any obvious breaks. The most common distinguishing mark of inline elements is a font change, but inline elements may present no visual distinction at all. Inline elements contain character data and possibly other inline elements, but they never contain block elements. They are used to mark up data. Some examples are: cross references, filenames, commands, options, substrings and superscripts, and glossary terms.

Block Elements - paragraph-level elements

The block elements occur immediately below the component and sectioning elements.

Lists

<table>
<thead>
<tr>
<th>List Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalloutList</td>
<td>A list of marks, frequently numbered and typically on a graphic or verbatim environment and their descriptions.</td>
</tr>
<tr>
<td>GlossList</td>
<td>A list of glossary terms and their definitions</td>
</tr>
<tr>
<td>ItemizedList</td>
<td>An unordered (bulleted) list</td>
</tr>
<tr>
<td>OrderedList</td>
<td>A numbered list</td>
</tr>
<tr>
<td>SegmentedList</td>
<td>A repeating set of named items. For example, a list of states and their capitals might be represented as a SegmentedList.</td>
</tr>
<tr>
<td>SimpleList</td>
<td>An unadorned list of items</td>
</tr>
<tr>
<td>VariableList</td>
<td>A list of terms and definitions or descriptions</td>
</tr>
</tbody>
</table>

Table 11: `<segmentedlist>` element, "statecapital.xml"

```xml
<segmentedlist>
  <title>State Capitals</title>
  <seglistitem><seg>Georgia</seg><seg>Atlanta</seg></seglistitem>
  <seglistitem><seg>Alaska</seg><seg>Juneau</seg></seglistitem>
  <seglistitem><seg>Arkansas</seg><seg>Little Rock</seg></seglistitem>
</segmentedlist>
```

Table 12: "statecapital.xml" output

The capitals of the states of the United States of America are:

**State Capitals**

State: Georgia

Capital: Atlanta

State: Alaska

Capital: Juneau

State: Arkansas

Capital: Little Rock

Table 13: `<orderedlist>` element, "mashpotatoe.xml"

```xml
<orderedlist numeration="upperroman">
  <listitem>
    <para>Preparation</para>
  </listitem>
  <listitem numeration="upperalpha">
    <listitem><para>Chop tomatoes</para></listitem>
    <listitem><para>Peel onions</para></listitem>
    <listitem><para>Mash potatoes</para></listitem>
  </listitem>
</orderedlist>
```
I. Preparation
A. Chop tomatoes
B. Peel onions
C. Mash potatoes

II. Cooking
A. Boil water
B. Put tomatoes and onions in
C. Blanch for 5 minutes

Admonitions

There are five types of admonitions: Caution, Important, Note, Tip, and Warning.

Line-specific environments

Line-specific environments preserve whitespace and line breaks.

<table>
<thead>
<tr>
<th>Address</th>
<th>A real-world address, generally a postal address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiteralLayout</td>
<td>A block of text in which line breaks and white space are to be reproduced faithfully</td>
</tr>
<tr>
<td>ProgramListing</td>
<td>A literal listing of all or part of a program</td>
</tr>
<tr>
<td>Screen</td>
<td>Text that a user sees or might see on a computer screen</td>
</tr>
<tr>
<td>ScreenShot</td>
<td>A representation of what the user sees or might see on a computer screen</td>
</tr>
<tr>
<td>Synopsis</td>
<td>A general-purpose element for representing the syntax of commands or functions</td>
</tr>
</tbody>
</table>

Common block-level elements

Common block-level elements include Examples, figures, and tables. The distinction between formal and informal elements is that formal elements have titles while informal ones do not.

Example, InformalExample

<table>
<thead>
<tr>
<th>Example</th>
<th>element</th>
</tr>
</thead>
</table>

Figure, InformalFigure

<table>
<thead>
<tr>
<th>Figure</th>
<th>element</th>
</tr>
</thead>
</table>
Table, InformalTable

Table 19: <table> element

Equations

Equation and InformalEquation (without titles)

Graphics

InlineGraphic, MediaObject, InlineMediaObject

These elements may contain video, audio, image, and text data. A single media object can contain several alternative forms from which the presentation system can select the most appropriate object.

Please see Table 18 for example.

Inline Elements – used to mark up running text

In published documents, inline elements often cause a font change or other small change, but they do not cause line or paragraph breaks.

<table>
<thead>
<tr>
<th>Abbrev</th>
<th>An abbreviation, especially one followed by a period.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acronym</td>
<td>An often pronounceable word made from the initial (or selected) letters of a name or phrase.</td>
</tr>
<tr>
<td>Emphasis</td>
<td>Emphasized text.</td>
</tr>
<tr>
<td>Footnote</td>
<td>A footnote. The location of the Footnote element identifies the location of the first reference to the footnote. Additional references to the same footnote can be inserted with FootnoteRef.</td>
</tr>
<tr>
<td>Phrase</td>
<td>A span of text.</td>
</tr>
<tr>
<td>Quote</td>
<td>An inline quotation.</td>
</tr>
<tr>
<td>Trademark</td>
<td>A trademark.</td>
</tr>
<tr>
<td>Citation</td>
<td>An inline bibliographic reference to another published work.</td>
</tr>
</tbody>
</table>
Publishing a DocBook Document

DSSSL vs XSL Stylesheets:

Document Style Semantics and Specification Language (DSSSL) is a stylesheet language for both print and online rendering. It is mainly intended to work with SGML.

Extensible Stylesheet Language (XSL) is a language for expressing stylesheets written in XML. It includes the formatting object language, but refers to separate documents for the transformation language and the path language. In this chapter, we will use XSL Stylesheets because they’re more powerful, you are already familiar with them, and they are intended to work with XML.

Step 1: Get the Standard StyleSheets

DocBook strictly separates the content and appearance of a document. A DocBook document only explains the semantics of the document, not its formatting or appearance. In order to publish your DocBook Document, you will need to use a set of DSSSL or XSL Stylesheets describing the formatting and an XSL processor.

If you’re thinking that it would be a lot of work to write your own XSL stylesheets, you’re right. The good news is that you don’t need to. There are a large number of freely available standard XSL stylesheets for DocBook maintained primarily by Norman Walsh.

Make sure that you download the latest version of these stylesheets at Sourceforge.net - Stylesheets Repository (http://sourceforge.net/project/showfiles.php?group_id=21935). The stylesheet distribution consists of a collection of modular XSL files that are assembled into several complete XSL stylesheets. There is a stylesheet for generating a single HTML file, and one for generating multiple smaller HTML files from a single DocBook document. There are stylesheets for print output, XHTML output, HTML Help output, and JavaHelp output. Since there are XSL processors for all major computer types, you can use DocBook on Unix, Linux, Windows, and Macintosh computers. By using these default stylesheets installed on your system, it is quite easy to create customized stylesheets. But don’t forget to note that the common approach to customize the stylesheets is creating a customization layer rather than editing them directly.

Step 2: Download an XSLT processor

To publish HTML from your XML documents, you will need an XSLT engine. To print, you need an XSLT engine to produce formatting objects (FO), which then must be processed with an FO engine to produce PostScript or PDF output. A variety of XSLT engines are available. Here’s a list of some free/open-source ones you might consider. Note that xsltproc and Saxon are currently the only recommended XSLT engines for use with DocBook.

XSLT Engines

1. Xsltproc: A free processor written in C, available as part of the open source libxml2 library from the Gnome development project. It is considered the fastest of the processors, and is highly conformant to the specification. Download at http://xmlsoft.org/XSLT/
2. Saxon: A free processor written in Java, that can be run on any operating system with a modern Java interpreter. It uses the Aelfred XML parser internally, which has some bugs, so many people substitute the Xerces parser. Download at http://saxon.sourceforge.net/
3. Xalan: Xalan is part of the Apache XML Project. It has versions written in both Java and C++, both of them free. The Java version is highly portable and more fully developed. Generally Xalan is used with the Xerces XML parser (Java or C++), also available from the Apache XML Project. Download at http://xml.apache.org/
Your choice of an XSLT engine may depend a lot on the environment in which you'll be running the engine. Many DocBook users who need or want a non-Java application use xsltproc. It's very fast and the developers respond very quickly to bug reports and questions. But one current limitation xsltproc has is that it doesn't yet support Norm Walsh's DocBook-specific XSLT extension functions.

Saxon is the most popular one for use in a Java environment. It also supports Norm Walsh's DocBook-specific XSLT extension functions.

NetBeans IDE, the XML editor we've been using for other chapters' exercises, has a built-in XSLT processor using the XALAN parser by default. NetBeans IDE not only lets you validate your XML documents but also does XSL transformations right there in the IDE. It will work for this chapter's purpose well enough. It doesn't provide any XSLT debugging though, so you might want to get yourself a decent XSL IDE (e.g., XML Spy or Xcelerator) for serious XSLT work.

FO Engines

For generating print/PDF output from FO files, there are two free/open-source FO engines:


### Step 3: Customize the XSL stylesheets

#### Output to HTML

- The main strength of standard stylesheets is that they are easily customizable.
- Parameters found in params.xml
- Call your customization layer instead of the standard stylesheet

Table 22: A customized XSL stylesheet, "myxsl1.xsl"

```xml
<?xml version="1.0"?>
<!-- Customization layer -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- Use 'chunk.xsl' in line below to chunk files. -->
  <xsl:param name="chapter.autolabel" select="1"/>
  <xsl:param name="section.autolabel" select="1"/>
  <xsl:param name="section.label.includes.component.label" select="1" doc:type="boolean"/>
  <!-- Insert more parameters here. -->
</xsl:stylesheet>
```

Beyond setting parameters, you can modify XSLT "templates" to override default behavior
- You need at least a minimal knowledge of XSLT

Table 23: A customized XSL stylesheet, "myxsl2.xsl"

```xml
<xsl:template match="emphasis">
  <!-- Use 'chunk.xsl' in line below to chunk files. -->
  <xsl:when test="(@role='strong') or (@role='bold')">
    <xsl:call-template name="inline.boldseq"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:call-template name="inline.italicseq"/>
  </xsl:otherwise>
</xsl:template>
```

#### Output to PDF

It generally requires a two-stage process:

1. Generation of FO from XML
2. Generation of PDF from FO

Table 24: An XSL stylesheets to generate FO, "myxsl3.xsl"

```
xsltproc -o sample.fo $DB/fo/docbook.xsl sample.xml
fop.sh -fo sample.fo -pdf sample.pdf
```

### Extensions

a. **Slides Doctype**
- Creation of presentation slides from DocBook XML
- You can create HTML (with or without frames) and FO
- Uses DocBook elements within a specific hierarchical framework
- Downloadable from DocBook Open Repository at SourceForge

Table 25: "slides.xml"

```xml
<!DOCTYPE slides SYSTEM "/usr/share/sgml/docbook/xsl-slides-1.1/slides.dtd">
<slides>
  <slidesinfo>
    <title>A Simple Approach to DocBook</title>
  </slidesinfo>
```

b. Website Doctype

- Creation of web sites from a collection of DocBook XML files
- Uses most DocBook elements within specific framework. It has separate files that control page navigation and hierarchy.
- Downloadable from DocBook Open Repository at SourceForge.

<table>
<thead>
<tr>
<th>Table 26: &quot;website.xml&quot;</th>
</tr>
</thead>
</table>

```xml
<DOCTYPE webpage SYSTEM "../website.dtd" [ 
<!NOTATION XML SYSTEM "xml">]
<!ENTITY test1a SYSTEM "test1a.xml" NDATA XML>
<!ENTITY test3 SYSTEM "test3.xml" NDATA XML>
<!ENTITY about.xml SYSTEM "about.xml" NDATA XML>>
<webpage id="home">
  <config param="desc" value="The Test Home Page"/>
  <config param="rcsdate" value="$Date: 2001/11/08 20:44:20 $"/>
  <config param="footer" value="about.html" altval="About..."/>
  <head>
    <title>Welcome to Website</title>
    <summary>Introduction</summary>
    <keywords>Rusen Gul, XSL, XML, DocBook, Website</keywords>
  </head>
  <para>This website demonstrates the DocBook.</para>
  <webtoc/>
  <section>
    <title>What is a Website?</title>
    <para>A website is a collection of pages organized, for the purposes of navigation, into one or more hierarchies. In Website, each page is a separate XML document</para>
  </section>
</webpage>
```

Why use DocBook?

This certainly looks like too much work, doesn’t it? You’re not wrong. Why do we bother to use DocBook then?

- It is portable! A document written in DocBook markup can be converted into HTML, PostScript, PDF, RTF, DVI, plain ASCII text easily and quickly without any expensive tools.
- It is flexible! It enables output to multiple formats, including HTML, PDF, Slides, and many others.
- It separates the content from format! DocBook is only concerned with the structure of a document. It frees the author from worrying about the formatting and layout of a document.
- It is easy to understand! Most of DocBook elements are self-explanatory.
- It handles large quantities of content. You can physically divide the document into different files and work on them separately and conveniently.
- It is free! There a lot of freely available open source tools used to work with DocBook.

DocBook is well suited to any collection of technical documentation that is regularly maintained and published. Multiple authors can contribute to a single document, and their content can easily be merged because all the authors are using a highly structured, standard markup language. Just one little point to keep in mind; because the formatting for DocBook documents is strictly accomplished by stylesheets, DocBook is not well matched to highly designed layout-driven content like magazines.

Setting up a DocBook system will certainly take some time and effort. The payoff will be an efficient, flexible, and inexpensive publishing system that is iterative and that can grow with your needs. Therefore, it is worth the effort!

DocBook Filters - Reading and Writing DocBook XML Using OpenOffice.org

The goal of the project is to use OpenOffice.org as a WYSIWYG editor of XML content to edit structured documents using styles. When exported, these styles are then transformed to XML tags. This section shows you how to enable and use DocBook filters. Below are some links to stylesheets that can be downloaded to use the latest transformations.

**Enabling the DocBook XSLT's in OpenOffice.org 1.1 Beta 2/RC**

There are three different ways to enable the DocBook filters.

1. Download the DocBook XSLT Stylesheets and OpenOffice.org Style Template
   - Using this method will make certain that the most recent stylesheets and OpenOffice.org style template will be used for import and export. It is required to download the following to import, export and modify DocBook documents in OpenOffice.org:
     - The relevant XSLT stylesheets for the XML transformations (All available here)
     - An OpenOffice.org style template that contains custom styles corresponding to DocBook tags (Available here)

   The most recent stylesheets support the import and export of DocBook documents with article or chapter as the top-level tag. The different stylesheets required for each of these operation are listed below:
OpenOffice.org Template required for Article and Chapter documents:

- DocBookTemplate.stw

Creating a new DocBook filter

- Go to Tools -> XML Filter Settings...
- Set Filter Name and Name of File Type to DocBook (Chapter)
- Go to the Transformation tab
- Set DocType to <chapter>
- For XSLT for Export browse to the chapter export stylesheet (sofftodocbookheadings_chapter.xsl).
- For XSLT for Import browse to the chapter import stylesheet (docbooktosoffheadings.xsl).
- For Template for Import browse to the style template (DocBookTemplate.stw).
- Click OK and close the XSLT Filter Setting dialog

To create a DocBook Article filter, the above steps can be repeated with article replacing chapter

1. Download the DocBook XSLT Jar Packages for Article or Chapter

   This method is more convenient, however there is no guarantee that the most recent stylesheets and OpenOffice.org template will be used.

1. Download the DocBook UNO component for Article only

   The DocBook UNO component adds filter support for the retention of unresolved XML entities.

   - Download the DocBookFilter
   - Unzip it to the <OOo install Dir>/
   - Run pkgchk in the <OOo install Dir>/program dir
   - The DocBook Article filter will now import DocBook unresolved entities as OpenOffice.org set variables

How to Import a DocBook document

A DocBook article or chapter document can now be opened using the File -> Open dialog.

- Go to File -> Open...
- Browse to the DocBook document.
- Click OK

The DocBook XSLT filter should automatically determine the root element of the document and import it with the matching XSLT filter. Alternatively, it is possible to browse manually to the desired DocBook filter in the File Type combo-box in the File -> Open dialog.

How to Export a DocBook document

The DocBook document can also be exported using the File -> Save As dialog.

- Go to File -> Save As...
- Browse to the location where the document is to be saved
- Click Save

Again, the DocBook XSLT filter should automatically determine the file type and export with the matching XSLT filter. Alternatively, it is possible to browse manually to the desired DocBook filter in the File Type combo-box in the File -> Save As dialog.

Using OpenOffice.org Headings and Styles for different DocBook tags

Using OpenOffice.org styles to represent DocBook tags The style template supplies all of the custom styles that are currently supported. Once a DocBook document has been imported to OpenOffice.org, the available DocBook specific styles can be viewed using the Stylist. On import, each of the supported DocBook tags will be mapped to formatted OpenOffice.org content. Similarly, to modify the imported DocBook document, OpenOffice.org text styles can be used to represent the DocBook tags marking-up the text. NOTE: A new DocBook document can be created in OpenOffice.org by opening the DocBookTemplate.stw. The document can then be saved as a DocBook document, and the new content will be represented as DocBook mark-up. How to create new DocBook content:

- Press F11 to display the Stylist
- Select Custom Styles in the Stylist combo-box
- Click the Character Styles icon (second from left on the Stylist)
- Double-click the SubScript style
- Enter text in the OpenOffice.org document
- On exporting as DocBook, the text formatted as the SubScript custom style will be marked-up with the DocBook tag <subscript>

How to create DocBook sections: Initially the DocBook project used OpenOffice.org sections to enforce the nesting of DocBook sections. Feedback has shown that authors wish to use the common word processing styles such as Heading1, Heading2, etc. The following instructions describe how to create a <sect1> that contains a <sect2>

- Press F11 to display the Stylist
- Select All Styles in the Stylist combo-box
- Click the Paragraph Styles icon (first in the left on the Stylist)
- Double-click the Heading 1 style
Exercises

1. Download the latest DocBook DTD at http://www.oasis-open.org/docbook/xml/. Convert the Learning Objectives part of Chapter 2 of this textbook into a DocBook document. Check that it is well-formed and valid.


References and Useful Links

1. Docbook Official Website (http://www.oasis-open.org/docbook/)
13. DocBook mailing list: mailto:docbook@lists.oasis-open.org

Introduction

With the explosion of the late 90's popularity of the internet, The World Wide Web Consortium (W3C) saw the need to extend the capabilities of the web with respect to information structure and media presentation. This is how they arrived at XML, the extensible language for describing information structure. Furthermore, SMIL is built upon XML: it is a specialized language to describe the presentation of media objects. Since the W3C (and everyone else) doesn't know what media types will be around in the future (virtual environments, brainwave-synch experiences, psychic/holographic/video), XML was an appropriate choice in designing SMIL to be extended to support these media.

In order to integrate this technology with HTML and extend the application of media in HTML, the W3C decided to make a push towards modularizing these languages or protocols. SMIL is one of many modular languages which 'plug-in' to the larger framework of XML.

What is SMIL?
When adding this new mime-type to a web browser, the definition will need to include the ‘smil’ extension. In order for a MIME user-agent to recognize SMIL 2.0 files, the user-agent needs to be defined:

- `application/xhtml+xml` [current MIME type]
- `application/xhtml+xml` [deprecated]
- `application/xhtml+xml` [MIME type for embedding smil in XHTML]

When adding this new mime-type to a web browser, the definition will need to include the ‘smil’ extension.
The following hyperlink will direct you to the SMIL 2.0 Schemas, provided by the W3C.org. The main schema is a general description of SMIL 2.0 modules. It is followed by each module's schema. The main schema contains the include statements for all of the module's schemas.

W3C.Org's SMIL Schema description (http://www.w3.org/TR/smil20/smil-SCHEMA.html/language)

### SMIL. Namespace Declarations

SMIL 2.0 files need to have the following namespace declaration in the beginning `<smil>` tag:

**SMIL 2.0 namespace**

```xml
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"/>
```

SMIL 1.0 files have the following namespace declaration:

**SMIL 1.0 namespace**

```xml
<smil xmlns="http://www.w3.org/TR/REC-smil">
```

If no default namespace is declared within the `<smil>` tag then the document will be processed as SMIL 1.0.

### SMIL Syntax

**Guidelines and Rules**

SMIL documents look a lot like HTML. SMIL files need to be written according to the following rules:

- SMIL tags are case sensitive.
- All SMIL tags are written with lowercase letters.
- SMIL documents must start with a `<smil>` tag and end with a `</smil>` closing tag.
- SMIL documents must contain a `<body>` tag for storing the contents of the presentation.
- SMIL documents can have a `<head>` element (like HTML) for storing metadata information about the document itself, as well as presentation layout information.

**SMIL template**

**SMIL 1.0 template**

```xml
<smil>
  <head>
    <!-- The layout section defines regions in which to place content -->
    <layout>...
      </layout>
    <!-- Transitions defined in head act on content defined in body -->
    <transition id="fade" type="fade" dur="1s"/>
    <transition id="push" type="pushWipe" dur="0.5s"/>
  </head>
  <body>
    <!-- The body section defines the content to be used and how it will be displayed -->
    <par>
      <img src="imagefile.jpg" transIn="fade"/>
      <video src="soundfile.aif" transOut="push"/>
    </par>
  </body>
</smil>
```

**A Simple SMIL**

**Abbreviated SMIL markup**

```xml
<smil version="1.0" encoding="ISO-8859-1">
  <smil xmlns="http://www.w3.org/SMIL20/Language">
    <head>
      <!-- The layout section defines regions in which to place content -->
      <layout>...
        </layout>
      <!-- Transitions defined in head act on content defined in body -->
      <transition id="fade" type="fade" dur="1s"/>
      <transition id="push" type="pushWipe" dur="0.5s"/>
    </head>
    <body>
      <!-- The body section defines the content to be used and how it will be displayed -->
      <par>
        <img src="imagefile.jpg" transIn="fade"/>
        <video src="soundfile.aif" transOut="push"/>
      </par>
    </body>
  </smil>
```

```
An example SMIL

Example SMIL which has in-line text and an image

```xml
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
</head>
<body>
  <layout>
    <root-layout>
      <region id="image_region" src="sample_jpg.jpg" width="320" height="240"/>
      <region id="text1_region" src="data:text/plain,First%20Slide" dur="3s"/>
      <region id="text2_region" src="data:text/plain,Second%20Slide" dur="2s"/>
      <region id="text3_region" src="sample_jpg.jpg" region="image_region" dur="3s"/>
    </root-layout>
  </layout>
  <head>
    <body>
      <text src="data:text/plain,First%20Slide"/>
      <text src="data:text/plain,Second%20Slide"/>
      <img src="sample_jpg.jpg" region="image_region"/>
    </body>
  </head>
</smil>
```

Note that when using in-line text instead of referring to separate plain-text files as the text source, you will have to encode the text for any non-alphanumeric characters. This example uses '%20' in lines 13,14,15 as a space character. Also note that in line 13 the source for the text content begins with 'data:text/plain'. In SMIL 2.0 this is the default mime-type for text sources, so specifying it here is optional. In SMIL 1.0, however, this would have to be specified in order to use inline text.

SMIL 2.0 Modules

SMIL 2.0 divides the language description by functionality into ten modules. Each module contains elements to describe structure, content, actions or attributes. The following 10 modules are associated with the SMIL 2.0 namespace.

1. Timing
2. Time Manipulations
3. Animation
4. Content Control
5. Layout
6. Linking
7. Media Objects
8. Metainformation
9. Structure
10. Transitions

The **timing** module provides a framework of elements to decide whether elements appear concurrently, in sequence, or out of order and called by interactive events such as clicking on a hyperlink.

The **time manipulations** module provides the ability to associate media objects with time-related information such as the length of time a media object should be displayed, and a description of the timeline used as a frame of reference for the timing module.

The **animation** module allows media objects to be placed on a timeline defined by the time manipulations module.

The **content control** module allows for choices of which content is played, depending on such things as language and playback capabilities, using tags such as switch present a test of the system's capabilities.

The **layout** module contains elements that describe the spacial placement of media objects in the presentation.

The **linking** module describes hyperlinks and linking references to media objects.

The **media objects** module describes the pathing and typing of media objects.

The **metainformation** module contains elements that describe meta information about the SMIL file itself or the media objects it contains.

The **structure** module is a framework to describe the structure of the SMIL file such as the head and body and SMIL elements.

The **transitions** module is a framework to describe transitions such as wiping and fading between the presentation of media objects.

Viewing a SMIL file

In order to view a SMIL presentation, a client will need to have a SMIL player installed on his/her computer. Currently, Apple's Quicktime player, Windows Media Player (WiMP) and RealNetworks RealPlayer are among the most popular media players.

It would be convenient to be able to show these SMIL files natively in web browser, eliminating the requirement of a separate SMIL player or plug-in. Currently, Microsoft's Internet Explorer has limited support for SMIL features. The open-source Mozilla project is slowly incorporating SMIL and other XML-related technologies such as SVG and MathML into their browsers, but progress is slow. It is possible they are waiting for these XML-based languages to mature.

Embedding SMIL files into XHTML web pages

As mentioned, SMIL is not yet native to web browsers, so in order to put SMIL in a web page, one must embed it and open it in a plug-in. Embedding SMIL files into web pages is somewhat beyond the scope of this chapter. However, should you have a need to do this, the following links are included as references to help you.

- Embedding a SMIL file is easy to do with Apple's Quicktime (http://developer.apple.com/documentation/QuickTime/IQ_InteractiveMovies/quicktimeandsmil/chapter_10_section_5.html) media player.
- Use the Windows Media Player to view SMIL files in a web page on a non-IE browser.
- The Internet Explorer 5.5+ browser has support for SMIL.
- Visit this W3Schools page (http://www.w3schools.com/smil/smil_html.asp) for details on how to use SMIL in IE-only web pages.
SMIL for phones

As mentioned, SMIL is often used in the latest cellular phones. Phones and vendors have varied support for MMS (multimedia messaging service), but generally, MMS uses SMIL to define the layout of multimedia content. If the MMS message contains a SMIL file, it will include other media objects, which can be text or binary (text is treated here as a media object or file to be referenced in a smil file).

Just a general note on MMS: the telecommunications industry needed a system in order to charge for messages by throughput as well as a system for pushing multimedia messages from phone to phone, computer to phone or phone to computer. MMS is a standard, international system for these purposes. SMIL was adopted because it was a well-defined, standard language to describe the layout and timing of the content inside MMS messages. In adhering to these (and other) standards developed by the 3GPP in partnership with the European Telecommunications Standards Institution (ETSI) and the W3C, the industry was able to ensure interoperability of new services between vendors, providing mutual benefit and equal opportunity.

SMIL tools and SMIL Info

Given that WikiBooks is a publicly-available 'open' book, it would be inappropriate to include information about or links to any commercial SMIL tools. In other words, everything that is not free or open source is not considered here.

Just a sidenote: some commercial tools cost upwards of $800. It is therefore in our best interest to evaluate, provide feedback for, and contribute to opensource projects.

The following are useful links (March 18th, 2004) to free and opensource tools, current SMIL projects, specifications, and tutorials:

- The official W3C SMIL page (http://www.w3.org/AudioVideo/)
- X-Smiles - (http://www.x-smiles.org/) a Java-based, "an open XML browser for exotic devices." Supports XSLT, XSLFO, SMIL, XForms and SVG.
- Ambulant's Open SMIL Player (http://www.cwi.nl/projects/Ambulant/)
- W3school's excellent tutorial (http://www.w3schools.com/smil/default.asp) on SMIL.
- PerlySMIL (http://www.webiphany.com/perlysmil/) is a perl script for generating SMIL files from perl.
- LimSee2 - (http://wam.inrialpes.fr/software/limsee2/) is an opensource; Java application for generating SMIL. It is this author's experience that several media-related Java dependencies must be properly installed before LimSee2 will work properly.

SMIL in netbeans?

One can create a SMIL file in Netbeans just as one would create an XML file. Just type it up and save it as a SMIL file. You can check for well-formedness, but validation might be trickier. As mentioned previously, SMIL 2.0 requires a namespace declaration, so don't forget it.

For our simple exercises, just type up a well-formed SMIL document and save it as .smil That's it!

Summary

We've seen how SMIL could be used to make standalone presentations. Yet the future of SMIL may be in the connection of mobile devices to the internet. As XML standards and SMIL tools reach maturity, SMIL will be increasingly implemented in order to define interactive presentations in the same way that Macromedia FLASH does, only this presentation will be native to web browsers and micro browsers used in mobile devices. Since SMIL is an open standard and it is extensible, there will likely be other applications which will use also SMIL.

Visionaries foresee the increasing ubiquity of the internet in our homes and work, on computers and mobile devices. This ubiquity is also called 'pervasive computing'. Mobile commerce would be an example of pervasive computing as cellular phones and portable devices become more useful for business and location-based services. SMIL is a language which facilitates this trend by providing either a pretty face for future business services or value-added multimedia content.

SMIL Exercises

1. Create a simple SMIL file which displays the words, 'Hello World'. Confirm that the file works in a SMIL-conformant player.
2. Author a SMIL file which displays 'Hello World' for 3 seconds, then displays 'Goodbye World' for 1 second. Confirm that the file works in a SMIL-conformant player.
3. Take an existing Openoffice.org present (or PowerPoint) presentation and turn it into a SMIL file. Double check it in a SMIL browser.
4. Embed one of the previously created SMIL files into an XHTML web page and store the SMIL file and web page on a server. Confirm that the SMIL file works for two different computers.

SMIL Answers

References


Learning objectives

Upon completion of this chapter, you will be able to answer the following questions:

- What is WDDX?
- What are the uses of WDDX?
- What are Web Syndicate Networks?
- What are the benefits of WDDX?

Introduction

WDDX (Web Distributed Data eXchange) was created by Allaire, now known as Macromedia (http://www.macromedia.com), to solve the problem of exchanging data between different web applications. It was originally intended to be used to exchange data between ColdFusion (http://www.macromedia.com/software/coldfusion/) and other web application languages. This XML-based technology enables complex data to be exchanged between totally different Web programming languages by creating 'Web Syndicate Networks.' WDDX consists of a language-independent representation of data based on an XML 1.0 DTD, and a set of modules for a wide variety of languages that use WDDX.

Features

- can be used with HTTP, SMTP, POP, FTP and other Internet protocols that support transferring textual data
- must be using Netscape 3.0+ or Internet Explorer for Windows or any versions for Unix and Mac platforms
- supports boolean, number, date-time, and string data types
- supports complex data types like arrays, structures, and recordsets
- not a formal standard, but is free and widely distributed and is based on standard-base technologies like XML 1.0

How it Works

The way that the web distributed data exchange works is by assigning a specific module for the given programming language to translate the data into an abstract XML format. Another specific module then translates the XML back into another programming language for another web application. For example, if you had an array in a ColdFusion program that you wanted to send to an ASP program, it would first be serialized into XML and sent to the ASP server. The ASP server would then deserialize it and convert it to VBScript for use in the ASP program.

Web Syndicate Networks

The term 'Web Syndicate Network' refers to a group of websites that share their content and transactions. This allows for economies of scale as each site can use shared database content or even transactions and procedures.

References

OpenWDDX - http://www.openwddx.org
WDDX Functions for PHP - http://www.php.net/wddx
Learning Objectives

Upon completion of this chapter, you will be able to

- Discuss XML-RPC
- Create XML-RPC Code based on Examples

RPC Defined

In order to fully understand XML-RPC, we should first define RPC. A Remote Procedure Call (RPC) is a protocol that allows a computer program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this. An RPC is initiated by the caller (client) sending a request message to a remote system (the server) to execute a certain procedure using arguments supplied. A result message is returned to the caller. There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols.

In order to allow servers to be accessed by differing clients, a number of standardized RPC systems have been created. Most of these use an Interface Description Language (IDL) to allow various platforms to call the RPC. Web services were the first real attempt to implement RPC between platforms. Using Web services a .NET client can call a remote procedure implemented in Java on a Unix server (and vice versa).

Web services use XML as the IDL, and HTTP as the network protocol. The advantage of this system is simplicity and standardization, the IDL is a text file that is widely understood, and HTTP is built into almost all modern operating systems. An example of such an RPC system is XML-RPC.

XML-RPC Defined

XML-RPC (Extensible Markup Language Remote Procedure Call) is a Remote Procedure Call protocol encoded in XML. It is a very simple protocol, defining only a handful of data types and commands, and the entire description can be printed on two pages of paper. This is in stark contrast to most RPC systems, where the standards documents often run into the thousands of pages and require considerable software support in order to be used.

It was first created by Dave Winer (http://en.wikipedia.org/wiki/Dave_Winer/) in 1995 with Microsoft (http://www.microsoft.com/). However, Microsoft considered it too simple and started adding functionality. After several rounds of this, the standard was no longer so simple and became what is now SOAP (http://en.wikipedia.org/wiki/SOAP/).

"We wanted a clean, extensible format that's very simple. It should be possible for an HTML coder to be able to look at a file containing an XML-RPC procedure call, understand what it's doing, and be able to modify it and have it work on the first or second try... We also wanted it to be an easy to implement protocol that could quickly be adapted to run in other environments or on other operating systems." - xmlrpc.com (http://www.xmlrpc.com/)

Data Types

Data Types Referenced from XML-RPC (http://en.wikipedia.org/wiki/XML-RPC/)
<table>
<thead>
<tr>
<th>name</th>
<th>tag example</th>
<th>description</th>
</tr>
</thead>
</table>
| array | ```<array>
<data>
  <value><i4>1404</i4></value>
  <value><string>Something Here</string></value>
  <value><i4>1</i4></value>
</data>
</array>``` | Array of values, storing no keys |
| base64 | `<base64>eW91IGNhbid0IHJlYWQgdGhpcyE=</base64>` | [Base 64]-encoded binary data |
| boolean | `<boolean>1</boolean>` | [Boolean] logical value (0 or 1) |
| date/time | `<dateTime.iso8601>19980717T14:08:55</dateTime.iso8601>` | Date and time |
| double | `<double>-12.53</double>` | Double [precision] floating number |
| integer | `<i4>42</i4>` | Whole number, [integer] |
| string | `<string>Hello world!</string>` | String of characters. Must follow XML encoding. |
| struct | ```<struct>
  <member>
    <name>foo</name>
    <value><i4>1</i4></value>
  </member>
  <member>
    <name>bar</name>
    <value><i4>2</i4></value>
  </member>
</struct>``` | Array of values, storing keys |
| nil | `<nil/>` | Discriminated null value; an XML-RPC extension (http://ontosys.com/xml-rpc/extensions.php) |

### Examples

An example of a typical XML-RPC request would be:

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

An example of a typical XML-RPC response would be:

```xml
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

A typical XML-RPC fault would be:

```xml
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

A final example, comparing a PHP associative array with an equivalent XML-RPC `<struct>`. This array:

```php
Array
[
    [0] => 'dogs',
    [1] => 'cats',
    ['animals'] => Array[
        [0] => FALSE,
        [1] => 'little_dogs',
        [2] => 'little_cats',
        [3] => 5,
        [4] => 2.3,
        [5] => 1,
    ],
]
```
<?xml version="1.0" encoding="utf-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>0</name>
            <value><string>dogs</string></value>
          </member>
          <member>
            <name>1</name>
            <value><string>cats</string></value>
          </member>
          <member>
            <name>animals</name>
            <value>
              <array>
                <data>
                  <value><boolean>0</boolean></value>
                  <value><string>little_dogs</string></value>
                  <value><string>little_cats</string></value>
                  <value><i4>5</i4></value>
                  <value><double>2.3</double></value>
                  <value><boolean>1</boolean></value>
                </data>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>

References

- XML-RPC Homepage (http://www.xmlrpc.com/)
- Wikipedia (http://en.wikipedia.org/)
- Free Online Dictionary of Computing (http://foldoc.doc.ic.ac.uk/foldoc/)
- Forum (http://groups.yahoo.com/group/xml-rpc/)
- Technology Reports (http://xml.coverpages.org/xml-rpc.html)
- Citations from CiteSeer (http://citeseer.org/cs?q=XML+and+RPC)

Learning objectives

Upon completion of this chapter, you will be able to

- understand the Resource Description Framework (RDF)
- use RDF to define metadata for web resources
- include standards like the Dublin Core for your description
- explore how Adobe is handling metadata
- create your own individual properties to expand your description

Introduction
The Resource Description Framework (RDF) is terminology used to encode, exchange and reuse metadata in the World Wide Web. Metadata, structured data about data, includes any important type of information about a resource such as author, title, creation date or language. A resource is everything that can be addressed with a Uniform Resource Identifier (URI) (http://www.w3.org/TR/NOTE-rdf-simple-intro). For example, a web page or a distinct type of document. RDF considers description as the act of making statements about the properties (attributes, characteristics) and inter-relationships of these resources. A framework is a common model to contain or manage the diverse information about a resource.

Why do we not use XML to describe things?

- XML is too flexible. There are too many ways to describe things. For example, the name of a person (see code example). Each of these XML documents would map into a different logical tree. However, a query like what is the name of person x, has to be independent of the choice of the tree. RDF is different because it has a standard way of interpreting XML-encoded descriptions of resources which converts into one logical tree and thereby covers all possible representations of a description.

```
<person name="Pete Maravich">

or

\{person
  \<name>Pete Maravich</name>
\}</person>
```

- XML documents follow a schema. The order of elements is restricted, and documents are not extensible without changing the schema. RDF allows to list information regardless of their order or appearance. RDF is also openly extensible. This means if one receives a description about something or someone, one can easily add information without being limited to following a schema. This is a great advantage, particularly for annotation and metadata applications. Besides that, it is intricate to retrieve any semantic meaning from an XML document without knowing the XML schema.

RDF is an application of XML that enforces the needed structural constraints to provide unambiguous methods of expressing semantics. XML syntax guarantees vendor independence, extensibility, validation and the ability to represent complex structures. RDF extends the general XML syntax and model to be specific for describing resources. Furthermore, RDF uses XML namespaces that allow to scope and uniquely identify a set of properties. With namespaces that point to URIs, one can generate globally unique names for its resources. Unique names need no context to qualify.

Brief History

RDF is a result of several metadata communities coming together to build a robust and flexible architecture for supporting metadata on the existing web. The first RDF specification was released 1997 by Ora Lassila and Ralph Swick. Based on that specification RDF interest groups were established in the following years and RDF became a W3C recommendation (W3C RDF (http://www.w3.org/RDF/)). The potential of RDF was soon realized and once its use is widespread the impacts will be tremendous. Ora Lassila said the following (W3C NOTE_1997-11-13 (http://www.w3.org/TR/NOTE-rdf-simple-intro)).

*Once the web has been sufficiently "populated" with rich metadata, what can we expect? First, searching on the web will become easier as search engines have more information available, and thus searching can be more focused. Doors will also be opened for automated software agents to roam the web, looking for information for us or transacting business on our behalf. The web of today, the vast unstructured mass of information, may in the future be transformed into something more manageable - and thus something far more useful.*

Purpose

Besides the human-readable display of metadata RDF is intended to enable the exchange of information between different applications without any loss of meaning. The effective use of metadata among applications, however, requires common conventions about semantics syntax, and structure. RDF imposes these conventions that make an unambiguous transfer possible. Application areas include resource description, site-maps, content rating, electronic commerce, collaborative services, and privacy preferences. Earlier one of the major obstacles of metadata interoperability has been the multiplicity of incompatible standards for metadata syntax and schema definition languages. However since RDF is a W3C recommendation and communities provide a standard vocabulary to describe things application designers and developers can create applications that allow metadata exchange in a standardized way.

The Basic Structure

Statements

With RDF one can make statements about resources. Below you can see an example of a statement that can be made about a web page. The key parts of the statement are highlighted:

http://www.example.org/index.html has an author whose name is Pete Maravich.

In general a RDF statement is a triple that contains a:

- Resource, the subject of a statement
- Property, the predicate of a statement
- Value, the object of a statement

RDF is based on the concept that every resources can have different properties which have values. A resources, represented by an URI reference, can be fully described by using properties and their values. Other properties for this web page could be:

http://www.example.org/index.html has a language which is English.

or

http://www.example.org/index.html has a title which is Example_Title.
Graphs

An RDF statement is a structured triple that contains a subject, a predicate and an object. A set of such triples is called a graph where a subject is always a node, a predicate is always an arc and an object is always a node:

![Figure 1] (Abstract RDF graph)

The set of example statements can be represented by the following graph:

![Figure 2] (Example graph)

<table>
<thead>
<tr>
<th>Subject</th>
<th><a href="http://www.example.org/index.html">http://www.example.org/index.html</a></th>
<th>is either an URI reference or a blank node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate</td>
<td><a href="http://purl.org/dc/elements/1.1%3ETitle">http://purl.org/dc/elements/1.1&gt;Title</a></td>
<td>is an URI reference</td>
</tr>
<tr>
<td>Object</td>
<td>Example_Title</td>
<td>can either be an URI reference, a literal or a blank node</td>
</tr>
</tbody>
</table>

RDF/XML

Natural English sentences and graphs that represent RDF's concept model are very useful practices to understand the basics of RDF. However RDF uses a normative XML syntax called RDF/XML to put down and exchange graphs. Like HTML, RDF/XML is machine processable and, using URIs, can link pieces of information. But, unlike conventional hypertext, RDF URIs can refer to any identifiable thing, including things that may not be directly retrievable on the Web (such as persons).

The following lines represent the graph in Figure 2 in RDF/XML:

```xml
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:property="http://www.example.org/properties/"
>
  <rdf:Description rdf:about="http://www.example.org/index.html">
    <property:author>Pete Maravich</property:author>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
    <dc:language>en</dc:language>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
    <dc:title>Example_Title</dc:title>
  </rdf:Description>
</rdf:RDF>
```

Figure 3 - example_rdf.rdf

Let's examine the lines of code to get a better understanding of the syntax:

1. XML declaration, identifies the document as XML in the current version 1.0
2. Start of an rdf:RDF element, identifies the following code as RDF - also declares an XML namespace rdf; all tags starting with the prefix rdf: are part of the namespace identified by the URIRef http://www.w3.org/1999/02/22-rdf-syntax-ns# which describe the RDF vocabulary
3. declares an XML namespace dc, all tags starting with the prefix dc: are part of the namespace identified by the URIRef http://purl.org/dc/elements/1.1/ - the link defines a standard vocabulary of terms for metadata
4. declares an XML namespace property, all tags starting with the prefix property: are part of the namespace identified by the URIRef http://www.example.org/properties/ - this URI is fictitious and was chosen to indicate that one can create their own vocabulary to describe resources
5. [5 to 7] represents a specific statement about the resource http://www.example.org/index.html as seen in the examples - Line 5 declares the subject of the description - Line 6 provides a property element, the qualified name property is an abbreviation that represents the assigned namespace (line 4), property:author stands for http://www.example.org/properties/author - embedded in the property tag is the value(object) of the description as a plain literal
6. [8 to 10] shows another statement - Line 8 again provides the subject - dc:language specifies the predicate for the statement, http://purl.org/dc/elements/1.1/language - the literal 'en' is an international standard two-letter code for English
7. [11 to 13] shows yet another statement - Line 10 to identify the subject - dc:title specifies the predicate for the statement, http://purl.org/dc/elements/1.1/title - the value Example_Title is the object
8. [14] ends the rdf:RDF element

Section 3 has covered the basic structure of RDF and is intended to provide a fundamental understanding of the topic. The next section will cover some advanced structures and features of RDF.

Advanced Concepts
As mentioned earlier the object of a statement can be a literal, a blank node or a URI reference. The latter two give RDF more power because they allow to create complex structures, so called \textbf{structured property values}. For instance you consider describing the address of somebody. An address is a structure that consists of different values such as a street, a city, a state and a zipcode. In RDF one would identify the address as a resource to allow a more detailed description.

As you can see the value of the property \textit{creator} is represented by a reference using the URI \url{http://www.example.org/members/1234}. RDF statements (additional arcs and nodes) can then be written with that node as the subject, to represent the additional information like the name of the creator and his address. The property \textit{address} itself is represented by a URI, which allows a detailed description that is aggregated from further statements about the address.

However the URIref \url{http://www.example.org/address/1234} may never need to be referred to directly from outside a particular graph, and therefore may not require a specific identifier. The concept above could also be represented by using a \textbf{blank node} for the address object. Blank nodes were called anonymous resources, they have no URIrefs and no literals.

In RDF/XML the concept of structured property values and blank nodes are represented like this:

```
<?xml version="1.0"?>
<br>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:property="http://www.example.org/properties/"
>
<br>
<rdf:Description
rdf:about="http://www.example.org/index.html"
>
<br>
<dc:creator>
<br>
<rdf:Description
rdf:about="http://www.example.org/members/1234"
>
<br>
<properties:name>Pete Maravich</properties:name>
<br>
<properties:address
rdf:nodeID="abc"/>
<br>
</rdf:Description>
<br>
</dc:creator>
<br>
</rdf:Description>
<br>
<rdf:Description rdf:about="http://www.example.org/members/1234"
>
<br>
<properties:name>Pete Maravich</properties:name>
<br>
<properties:address rdf:nodeID="abc"/>
<br>
</rdf:Description>
<br>
</dc:creator>
<br>
</rdf:Description>
<br>
<rdf:Description rdf:nodeID="abc">
<br>
<properties:street>346 Broad Street</properties:street>
<br>
<properties:city>Athens</properties:city>
<br>
<properties:state>Georgia</properties:state>
<br>
</rdf:Description>
<br>
</rdf:RDF>
```

Let's examine the lines of code that represent the new concepts:

- [5 to 12] describes the resource \url{http://www.example.org/index.html} that has the value \url{http://www.example.org/members/1234}
- [7 to 10] displays a way to abbreviate multiple property elements for a resource - usually a node has multiple arcs(properties) coming off and instead of writing one description for each property one can abbreviate this by using multiple child property elements inside the node element describing the subject node
- [9] shows how one can identify a blank node in RDF/XML - it is sometimes necessary that the same blank node in a graph is referred to in the RDF/XML in multiple places - if so, a blank node identifier can be given to the blank node for identifying it in the document
- [13-17] displays the properties and values for the blank node identified in line [9]

\section*{RDF applications}
Dublin Core Metadata Initiative

The Dublin Core Metadata Initiative (DCMI (http://dublincore.org)) is an organization dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems. Basically the Dublin Core is a set of elements that are used to describe a document. The goal of the Dublin Core is to provide a minimal set of descriptive elements that support and simplify the description and the automated indexing of document-like networked objects. Discovery tools on the Internet, such as the "Webcrawlers" employed by popular World Wide Web search engines use the metadata set. In addition, the Dublin Core is meant to be sufficiently simple to be understood and used by the wide range of authors and casual publishers who contribute information to the Internet. (also see RFC-2413 (http://www.ietf.org/rfc/rfc2413.txt))

Dublin Core Metadata Element Set (http://dublincore.org/documents/dces/) shows a description of all current elements defined in the Dublin Core. In the following example one can see an RDF document that uses the Dublin Core elements to describe an article in a magazine:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:mitglied="http://mitglied.lycos.de/virtuoso5/xmp/xmp.txt"
>
  <rdf:Description rdf:about="http://www.cio.com/archive/101504/km.html">
    <dc:title>A Less for Success</dc:title>
    <dc:creator>Alice Dragoon</dc:creator>
    <dc:description>Forget the big bang approach. When it comes to demonstrating the value of knowledge management, a piecemeal strategy works best.</dc:description>
    <dc:creator>CXO Media</dc:creator>
    <dc:date>October 15, 2004</dc:date>
    <dc:format>text/html</dc:format>
    <dc:language>en</dc:language>
  </rdf:Description>
</rdf:RDF>
```

Figure 7  -  cio_article.rdf

Adobe's XMP

The Extensible Metadata Platform is a specification describing RDF-based data and storage models for metadata about documents in any format. XMP can be included in text files such as HTML or SVG, image formats such as JPEG or GIF and Adobe's own formats like Photoshop or Acrobat. Adobe is making efforts that all of their applications will support XMP. However Adobe claims that XMP provides a standard format for the creation, processing and interchange of metadata, the specification is not a standard.

XMP provides the following:

- A data model - as a useful and flexible way of describing metadata in documents.
- A storage model - for the implementation of the data model. This includes the serialization of the metadata as a stream of XML and XMP Packets, a means of packaging the data in files.
- Schemas - predefined sets of metadata property definitions that are relevant for a wide range of applications, including all of Adobe’s editing and publishing products, as well as for applications from a wide variety of vendors. XMP also provides guidelines for the extension and addition of schemas.

However XMP metadata vocabularily is relatively small,i.e. the ways to describe a document are limited. To overcome this issue Adobe is using metadata standards such as the Dublin Core and also allows users to define their own metadata vocabularily.

The following screenshot is from Acrobat Professional 6.0 Document Metadata feature. The description field allows users to define metadata, whereas the advanced tab shows an overview of the metadata. Under view Source one can see the metadata in RDF/XML.

Adobe XMP example

Figure 8  -  Adobe XMP example

The RDF/XML based representation of the document's metadata can be found here (http://mitglied.lycos.de/virtuoso5/xmp/xmp.txt). The property funFactor expresses the hilariousness of a document. It was included using the 'load' functionality of Acrobat Professional to test the addition of arbitrary metadata to the properties Acrobat Professional already knew about.

RSS - RDF Site Summary

RSS Site Summary (RSS) is also an application of RDF. Please have a look at the Chapter on RSS in this Wikibook.

Creating an RDF Vocabulary

As seen earlier in the chapter one can create its own RDF metadata vocabulary, despite using standards like the Dublin Core. This section is intended to show a very general approach in creating such a personal vocabulary. For a detailed description please see Practical RDF - Powers 2003.

The first step in creating a vocabulary is to define the domain elements and their properties within the given area interest. This means one has to outline what kind of information about the resource should be described. Let's say we want to save the following facts about a resource:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Title of the resource</td>
</tr>
<tr>
<td>Created</td>
<td>Date of creation</td>
</tr>
<tr>
<td>Author</td>
<td>Author of the resource</td>
</tr>
<tr>
<td>Status</td>
<td>Current status of the resource</td>
</tr>
</tbody>
</table>
The next step is to create a RDF Schema (RDFS) document for the new vocabulary:

Here (http://mitglied.lycos.de/virtuoso5/examples/1.0/myschema) you can see the definition for our desired properties. Using this RDFS one can describe the article seen above the following way:

```xml
<?xml version="1.0"?><br/>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:same="http://mitglied.lycos.de/virtuoso5/elements/1.0/myschema#"><br/>
</rdf:Description><br/>
</rdf:RDF>
```

Figure 9 - cio_article2.rdf

To validate its RDFS and RDF files one can use the W3C RDF Validator (http://www.w3.org/RDF/Validator/).

Exercises

1. Create an RDF/XML document that describes an article of your choice (e.g. from magazines like CIO.com or ZDNet.com). Use the Dublin Core element set (http://dublincore.org/documents/dces/) and the Dublin Core terms (http://purl.org/dc/terms/) as a framework for your description. After completing the document please validate your work with the W3C RDF Validator (http://www.w3.org/RDF/Validator/).

References

- Shelley Powers - Practical RDF, O'Reilly 2003
- Dublin Core Metadata Initiative - http://dublincore.org/

Learning Objectives

Upon completion of this chapter, you will

- Understand the basics of RSS
- Understand the history of RSS
- Be able to construct a RSS 2.0 document using XML
- Subscribe to an RSS aggregator/reader

Introduction

RSS is a simple XML format used to syndicate headlines. It is now popularly used by websites that publish new content regularly and provide a list of headlines with links to their latest content. Content such as news feeds, events listings, project updates, blogger and most recently podcasting, video and image distribution can all be distributed by RSS. RSS feeds are also used by major Internet portals such as Google, Yahoo and AOL for people to personalize and have information that they care about delivered to them, i.e. MyYahoo.
RSS is considered a name variously used to refer to three different standards. The three separate branches are the RSS 0.9 branch, the RSS 1.0 branch (which is based on RDF) and RSS 2.0, and the initials have been expanded into three different names: "Really Simple Syndication" (RSS 0.9, 2.0), "Rich Site Summary" and "RDF Site Summary" (for RSS 1.0).

Several different versions have been developed by different developers under different names. According to XML.com, seven versions of RSS have been developed (see What is RSS? (http://www.xml.com/lpt/a/2002/12/18/dive-into-xml.html/)). Because RSS is understood as a term referring to many types of syndication protocols, these various RSS protocols have sometimes been accused of being "incompatible" with each other (see The myth of RSS compatibility (http://diveintomark.org/archives/2004/02/04/incompatible-rss)). This is an important issue for RSS reader/aggregator developers.

### History

The original version (version 0.90) of RSS was released by Netscape in 1999. Netscape developers were designing a format for making portals of headlines for news sites. After Netscape released the simplified version of RSS, they lost interest in developing RSS. However, another company, UserLand Software took over with intention to use RSS with their web-logging products and web-based writing software. While UserLand Software continued development with version 0.91, a third non-commercial group split off from the company and designed a new format based on version 0.90, which was a non-simplified version. The new format developed by this non-commercial group became known as version 1.0. In the meantime UserLand Software grew angered at the new 1.0 version, kept developing RSS and released version 2.0. Version 2.0 has become the leader and most widely adopted version of RSS. The 2.0 specification was donated to a non-commercial third party, Harvard Law School. Harvard Law is now responsible for the future development of the RSS 2.0 specification. Below is a table that describes each version, the owner, pros and cons, as well as its current status and recommendation for use.

<table>
<thead>
<tr>
<th>Version</th>
<th>Owner</th>
<th>Pros</th>
<th>Status</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.90</td>
<td>Netscape</td>
<td>Stable core, active module</td>
<td>Obsoleted by 1.0</td>
<td>Don't use</td>
</tr>
<tr>
<td>0.91</td>
<td>UserLand</td>
<td>Drop dead simple</td>
<td>Officially obsoleted by 2.0, but still quite popular</td>
<td>Use for basic syndication. Easy migration path to 2.0 if you need more flexibility</td>
</tr>
<tr>
<td>0.92, 0.93, 0.94</td>
<td>UserLand</td>
<td>Allows richer metadata than 0.91</td>
<td>Obsoleted by 2.0</td>
<td>Use 2.0 instead</td>
</tr>
<tr>
<td>1.0</td>
<td>RSS-DEV Working Group</td>
<td>RDF-based, extensibility via modules, not controlled by a single vendor</td>
<td>Stable core, active module development</td>
<td>Use for RDF-based applications or if you need advanced RDF-specific modules</td>
</tr>
<tr>
<td>2.0</td>
<td>UserLand</td>
<td>Extensibility via modules, easy migration path from 0.9x branch</td>
<td>Stable core, active module development</td>
<td>Use for general-purpose, metadata-rich syndication</td>
</tr>
</tbody>
</table>

### RSS structure

A RSS document is often known as RSS feed and can have three different types of file extensions: .RSS, .XML and .RDF. All RSS documents must conform 100% to the XML specification begin with the XML declaration. To identify a RSS document, the top level starts with a mandatory <channel> element which contains a brief description of the channel. Below is a sample of RSS(2.0) from the New York Times.

#### Exhibit 1: Data model for RSS

```xml
<?xml version="1.0" encoding="en-us" ?>

<rss version="2.0">
  <channel>
    <title>New York Times Home Page</title>
    <link>http://www.nytimes.com/index.html</link>
    <copyright>Copyright 2004 The New York Times Company</copyright>
    <language>en-us</language>
    <lastBuildDate>Sun, 7 Nov 2004 13:30:01 EST</lastBuildDate>
    <citie>
      <title>Iraq Declares State of Emergency as Insurgents Step Up Attacks</title>
      <description>Today`s attacks, including three police post raids that killed 21, came a day after insurgents killed at least 30. </description>
      <author>By EDWARD GOREY</author>
      <pubDate>Sun, 07 Nov 2004 00:00:00 EDT</pubDate>
    </citie>
  </channel>
</rss>
```

Figure 1-1: New York Times - HomePage.xml (http://www.nytimes.com/services/xml/rss/nYT/HomePage.xml) - RSS version 2

The `<channel>` element has three mandatory elements and several optional elements.

**Mandatory `<channel>` elements:**

- `<title>`: Description of the channel. Example: "The New York Times"

**Optional `<channel>` elements:**

- `<language>`: Description: Channel language. Example: en-us
- `<copyright>`: Description: Copyright notice for content in the channel. Example: Copyright 2004 The New York Times Company
- `<lastBuildDate>`: Description: The last time the content of the channel was updated/changed. Example: Sun, 7 Nov 2004 13:30:01 EST
Other optional elements include: managingEditor, webMaster, pubDate, category, generator, docs, cloud, ttl, image, rating, textInput, skipHours, skipDates. The requirement or sub-elements of each element please refer to the RSS specification (see at Harvard Law (http://blogs.law.harvard.edu/tech/rss/#hrelementsOfLtitemgt/Technology)). Below are example of image element.

**<image> elements:**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;link&gt;</td>
<td>The URL to the item</td>
<td><a href="http://www.nytimes.com/index.html">http://www.nytimes.com/index.html</a></td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>Picture title</td>
<td>NYT &gt; Home Page</td>
</tr>
<tr>
<td>&lt;url&gt;</td>
<td>The URL to the picture</td>
<td><a href="http://www.nytimes.com/images/section/NytSectionHeader.gif">http://www.nytimes.com/images/section/NytSectionHeader.gif</a></td>
</tr>
</tbody>
</table>

A channel may contain a number of &lt;item&gt;. An item may represent a "story" - much like a story in a newspaper or magazine; if so, its description is a synopsis of the story. The link points to the full story. An item may also be complete in itself, if so, the description contains the text (entity-encoded HTML is allowed; see examples), and the link and title may be omitted.

Each RSS channel can contain up to 15 items. All elements of an item are optional, however, an &lt;item&gt; element must contain at least one &lt;title&gt; or &lt;description&gt; element.

**&lt;item&gt; elements:**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;title&gt;</td>
<td>Title of the item</td>
<td>Iraq Declares State of Emergency as Insurgents Step Up Attacks</td>
</tr>
<tr>
<td>&lt;link&gt;</td>
<td>The URL to the item</td>
<td><a href="http://www.nytimes.com/2004/11/07/international/middleeast/07cnd-iraq.html">http://www.nytimes.com/2004/11/07/international/middleeast/07cnd-iraq.html</a></td>
</tr>
<tr>
<td>&lt;description&gt;</td>
<td>Brief description of the item</td>
<td>Today's attacks, including three police post raids that killed 21, came a day after insurgents killed at least 30.</td>
</tr>
<tr>
<td>&lt;author&gt;</td>
<td>Author's name and/or author's email address</td>
<td><a href="mailto:mail@nytimes.com">mail@nytimes.com</a> (Edward Wong)</td>
</tr>
<tr>
<td>&lt;pubDate&gt;</td>
<td>Date/time the item was published</td>
<td>Sun, 07 Nov 2004 00:00:00 EDT</td>
</tr>
<tr>
<td>&lt;guid&gt;</td>
<td>Is a string that uniquely identifies the item. Can be used by the aggregator to determine if an item is new.</td>
<td><a href="http://www.nytimes.com/2004/11/07/international/middleeast/07cnd-iraq.html">http://www.nytimes.com/2004/11/07/international/middleeast/07cnd-iraq.html</a></td>
</tr>
</tbody>
</table>

Others include: source, enclosure, category, and comments. (see at Harvard Law (http://blogs.law.harvard.edu/tech/rss/#hrelementsOfLtitemgt/Technology)).

An item can either be a child or a sibling of a channel.

**"child"**

```
&lt;channel&gt;
  ...
  &lt;item&gt;
    ....
  &lt;/item&gt;
&lt;/channel&gt;
```

**"sibling"**

```
&lt;channel&gt;
  ...
&lt;/channel&gt;

&lt;channel&gt;
  ...
  &lt;item&gt;
    ....
  &lt;/item&gt;
&lt;/channel&gt;
```

More optional elements visit RSS 2.0 Specification (http://blogs.law.harvard.edu/tech/rss/#optionalChannelElements)

**How does it work?**

RSS can be divided into two parts; the reader/ag and the feed. The reader is the program that reads and presents the RSS feed in an understandable format. The feed is the website with its RSS file. RSS feeds are typically identified on webpages with an orange rectangle icon, or an orange icon with the letters RSS written on it. To view the XML code, you simply have to click on the icon.

**Creating an RSS feed**

A website author can establish a RSS feed for itself in different ways; either by doing it manually, by using software or by online services. Most large websites use content management software to produce their RSS feed. Every time a change is made on their website, the content management software produce a RSS file of the changes with the new items added and old items removed.

**Subscribing to an RSS feed**

As a RSS subscriber you need a RSS aggregator. By feeding a RSS link, the aggregator will search for information you subscribed and display them. Say that you subscribe on the sport section in the New York Times; each time the NY Times publish a new sport article the article's headlines, description and the URL will be displayed on your computer. Whenever you are online, the aggregator will search out and sort your list of interests and display them.

**RSS Aggregators**

RSS aggregator (aka RSS Reader) is an application that is used to collect, update and display RSS feeds. Below is a list of some RSS aggregators for different platforms that the aggregator will work properly on.

- FeedReader (http://www.feedreader.com/) - Windows
- Sharp Reader (http://www.hutteman.com/weblog/cat_sharpreader.html/) - Windows(.NET)
- NetNewsWire (http://ranchero.com/netnewswire/) - Macintosh
- Straw (http://www.gnome.org/projects/straw/) - Linux
Some others include:

- Bloglines (http://www.bloglines.com/) - Server-Based
- NewsHutch (http://newshutch.com/) - Server-Based
- Radio Userland (http://radio.userland.com/newsaggregator/) - Windows, Macintosh
- PocketFeed (http://www.furrygoat.com/Software/PocketFeed/index.html/) - PocketPC

**Future of RSS**

The future of RSS seems very promising as version 2.0 has become extremely popular with the Internet industry and somewhat the standard of the RSS versions. Yahoo recently released its new version of Yahoo Maps and the API is based on georRSS version 2.0. This version of Yahoo Maps allows users to edit the information on the maps, which makes the Maps and Local Search products more effective. RSS version 2.0 is also very popular with distributing podcasts to the subscriber base along with distributing content Google’s blogger product. Furthermore, RSS is being utilized in an innovative way for search engine marketers to submit time sensitive content to the engines. The Mozilla Firefox browser already contains an internal RSS aggregator that allows users to view RSS news and blog headlines in the bookmark toolbar or bookmark menu. This is accomplished through the Mozilla Firefox feature named “Live Bookmarks”. RSS has quickly become a mainstream technology in a relatively short period and has definitely become a major player in the Internet space.

**Use RSS on Firefox**

**Summary**

Now, RSS is commonly used in areas such as, websites and blogs, with version 2.0 being the most popular standard. RSS feeds are typically identified on webpages with an orange rectangle icon, or an orange icon with the letters RSS written on it. To view the XML code, you simply have to click on the icon.

**Exercises**

**Answers**

**References**

Technology at Harvard Law - Internet technology hosted by Berkman Center - RSS 2.0 Specification (http://blogs.law.harvard.edu/tech/rss)
Mozilla Firefox - Live Bookmarks (http://www.mozilla.org/products/firefox/live-bookmarks/)
Apple - PodCasting (http://www.apple.com/itunes/podcasts/)
RSS INFO - RSS info (http://blogspace.com/rss/)
USA Today - USA Today (http://www.usatoday.com/)

**Learning objectives**

Upon completion of this chapter, you will be able to

- build a desktop client for a J2EE network service using JDNC

**Introduction**
Learning Objectives

Upon completion of this chapter, you will:

- be able to understand what XML Namespace is and its purpose
- be able to recognize XML Namespace structure and what each part is doing
- be able to think of organizations in which Namespace would be necessary

What is Namespace?

An XML namespace is a collection of names that are identified by a Uniform Resource Identifier (URI) reference, which are used in XML documents as element types and attribute names. URIs were used simply because they are a well-known system for creating unique identifiers. Namespaces consist of several parts including local names, namespace URIs, prefixes and declarations. The combination of a local name and a namespace is called a universal name. You might find it easier to think of a namespace as a dictionary that is a source of definitions for items that you use within an XML document.

All schemas include the namespace `http://www.w3.org/2001/XMLSchema-instance`. You can think of this as the master dictionary to which all schemas must refer because it defines the fundamental items of an XML schema. The namespace's address looks like a URL, but in XML we use the broader term Uniform Resource Identifier (URI).

Because a document can refer to multiple namespace, we need a convenient short form for referencing the namespace. One of the common forms used is `xsd` as illustrated in the following.

```xml
xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
```

The `xmlns` informs XML that you are referencing a name space, and the `xsd` indicates this is the short form of the namespace.

For example, you might use the following line of code in an XML schema

```xml
<xsd:element name="item" type="xsd:string" />
```

The previous line of code states that the definition of element name and string are found in "http://www.w3.org/2001/XMLSchema-instance"

Namespace enables you to use elements described in multiple schemas within your XML document, so the short form of a namespace's URI is useful for identifying the namespace to which you are referring.

History

Namespace in XML was a new W3C recommendation in January, 1999. Namespace was created to be a pretty simple method to distinguish names used in XML documents. The main purpose of Namespace is to provide programmers a method for which to grab elements and attributes that they want, leaving behind other tags that they do not need. These programmer-friendly names will be unique across the Internet. The XML namespaces recommendation does not define anything except a two-part naming system for element types and attributes.

For additional information regarding the W3C recommendation, follow this link: http://www.w3.org/TR/REC-xml-names/.

When would you use Namespace?

It would mainly be used to avoid naming conflicts. If you don’t have any duplicate elements or attributes in the XML that you use, namespaces are not necessary. It is however beneficial if you have duplicate elements or attributes. It basically makes two part structures that make it unique. Instead of just defining element A, for example, you have to define element A with some other type of identifier. That is where the URI comes into play. The URI in combination with the element or attribute creates your namespace and it is then a universal name.

Namespace Structure

XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.
The first declaration associates the `addr` prefix with the “www.example.com/addresses” URI. The second declaration defines `www.example.com/files` as the default namespace. If there is not a prefix defined for that element, a default namespace is applied. This default namespace is applied to all elements without a prefix. Please note, however, that default namespaces do not apply directly to attributes.

How Does It Work?

When specifying a universal name in an XML document, you use an abbreviation based on an optional prefix that's attached to the local name. This abbreviation is called the qualified name or qname. To declare an XML namespace, you use an attribute whose name has the form:

```
xmldnt:prefix
```

These attributes are often called xmlns attributes and their value is the name of the XML namespace being declared. This is a Uniform Resource Identifier. The first form of the attribute (xmlns:prefix) declares a prefix to be associated with the XML namespace. The second form (xmlns) declares that the specified namespace is the default XML namespace.

Namespace Best Practices

- Try to limit the number of Namespaces to about 5 per document. More than five namespaces in a document gets unwieldy.
- Make distinctions in XML namespaces only when there are truly distinctions between the things being named.
- Try to stick to documents in namespace normal form wherever possible because they are simplest to read and to process.
- Avoid overriding namespaces frequently because it can cause confusion in your documents.

Example of Namespace Use

Let’s say we are going to be pulling address values from two different sources and address from one source pulls in a mailing address while from the other source, it pulls in a computer IP address. We’ll need to create a Namespace so that we can distinguish the two addresses elements.

Postal Address XML document

```
<address>100 Elm St., Apt#1</address>
```

IP Address XML document

```
<address>172.13.5.7</address>
```

How do we distinguish these Address elements in the case that they need to be combined into the same document? We would assign each address name to a namespace. Therefore, it becomes defined in two parts, the address element and the XML namespace. Every time the element Address comes up, it will have to look at two things instead of one for definition, but this look up only has to be performed one time because the combination is universally unique.

In this instance, we could create Namespaces for the address element:

```
<Example Organization
xmlns:addr="http://www.example.com/postal_addresses"
xmlns="http://www.example.com/ip_addresses">
```

The first declaration associates the prefix 'addr' with the URI, "www.example.com/postal_addresses and the second declaration sets 'www.example.com/ip_addresses' as the default namespace. So, where a the prefix 'addr' is used, it will pull the postal address and for others, it will pull the IP address.

Defining the location of an XML schema

Assume you have created a schema, example.xsd, that is located in the same directory as your XML document, example.xml. In the XML document you will indicate the location of the schema with the following code.

```
<xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='example.xsd'>
```

Of course, if example.xsd is stored somewhere other than the same directory as example.xml, you specify the full path.

Potential Problems with Namespace

- Different XML technologies are going to process namespaces differently. Some will see namespace declarations as such and some will just see them as attributes.
- Namespace is a compromise solution that doesn't meet the needs of all users.
- XML namespaces seem simple on their face, but they can cause real confusion and increased complexity if they are not handled or managed correctly. To manage Namespaces correctly, you must understand thoroughly the meaning, rules, and implications of the various concepts that make up the XML namespaces mechanism and stick consistently to simple conventions.
Learning objectives

Upon completion of this chapter, for a single entity you will be able to

- create a report specification entirely in XML for Cognos ReportNet
- update a report specification in XML format.
- identify four main sections in a report specification

Introduction

Every report created in Cognos ReportNet has a specification that is written in XML, so you can customize it using XML editor or create a report specification entirely in XML.

Report Specification Flow

After you save a report and open it again, the report specification is pulled from the content store as you can see in Figure 28.1. When you edit it, the changes remain local on the client machine until you save it. When you save the report, the content store is updated.

Figure 28.1 Report Specification Flow

You can see a sample of web report in figure 28.2 and this report can be generated from XML file;

Figure 28.2 Sample of a report

XML in Report Specification Structure

A report specification consists of four main sections.

- Report Section
  - XML Tag: `<report>`
  - `<modelConnection>`
- Query Section
  - XML Tag: `<querySet>`
- Layout Section
  - XML Tag: `<layoutList>`
- Variable Section
  - XML Tag: `<variableList>`

At minimum, a report specification must include the `<report>`/`<report>` tags, as well as authoring language and schema information.

The specification header in Report Section includes information about:

- authoring language, “en-us” indicates American English. You can use other language than English for the report
- namespace : http://developer.sample.com/schemas/report/1
- package name: GSR
- model version : @name='model'
The query section includes information about:

- Cube elements are indicated by the `<cube>` tags which can contain:
  - facts (<factList> tags, Country, First Name and Last Name are the facts.
  - dimensions (<dimension> tags, consisting of levels (<level> tags)
  - filters (<filter> tags, consisting of conditions (<condition> tags)). Country is the filter for this report, which is equal to Germany.
- Tabular model is contained in the `<tabularModel>` tags.
  - Each tabular model contains data items (<dataItem> tags) consisting of fully qualified expressions (<expression> tags).
- The query section of a report is contained in the `<querySet>` tags.
- The query section can include multiple queries, each of which is contained in the `<BIQuery>` tags.

Add pages to a report specification:

- You can add many pages to a report. Each page is outlined between the `<pageSet>` tags.
  - Each page can consist of:
    - a body (mandatory)
    - a header
    - a footer

Add layout objects to a report:

- Once you have added one or more pages to the report layout, you can add a variety of layout objects, such as:
  - Text items
  - Blocks
  - Lists
  - Charts
  - Crosstabs
  - Tables

Specify styles for layout objects:

- You can use Cascading Style Sheets (CSS) attributes to determine the look and feel of objects in the layout.
- CSS values are specified between the `<style>` tags.
- CSS values can apply to things like font sizes, background colors, and so forth.

Add Variables to a Report:

- You can specify variables between the `<variableList>` tags of the report specification, and each variable includes an expression between `<expression>` tags.
- We can use Variable 1 that contains a list of possible values, example value: `fr` for using French language.

Below is the complete XML file for the report in Figure 28.3
Section summary: As Report Specification sticks to XML Rules it is favored for creating and updating a markup file.
Exercise

The end user wants to read the report in Japanese language, so you have to add a variable for Japanese language.

Answers

---

**Introduction**

MySQL is an open source relational database that supports XML. You can use the MySQL command line or a programming language of your choice to convert your MySQL databases and or tables to a well formed XML document.

**Supported Versions**

XML is supported in MySQL version 3.23.48 and higher. A free version of MySQL can be downloaded from MySQL.com (http://www.MySQL.com).

**Using the MySQL Command Line**

Use the `--xml` or `-X` option with either the mysqldump or mysql command to produce XML output.

**mysqldump Syntax:**

```
mysqldump --xml -u username -p databasename [tablename] > filename.xml
```

**mysql Syntax:**

```
\T "filename.xml"
```

**OR**

```
\T "filename.xml"
```

```
\T "filename.xml"
```

```
\T "filename.xml"
```

```
\T "filename.xml"
```

In the latter mysql syntax example, you can also specify a where condition as well as restrict the where condition just as you would in a regular sql select statement.

**Explanation of commands and options:**

- `mysqldump` is a mysql output command.
- `\T` is a mysql output command.
- `-e` is a mysql option that tells mysql to execute the following select statement.
- `--xml` is the mysql option for producing XML output.
- `-u` is a mysql option which tells mysql that the next command line item is your username.
- `username` is your mysql username. It will be used to authenticate you to the mysql database.
- `-p` is a mysqldump option that tells mysql that the next command line item is your password. If do not want your password to be visible on the command line, then do not supply your password after the `-p` option and mysql will prompt you for it later.
**databasename** is the name of the database that you want to output to XML. Supplying the tablename is optional.

**tablename** is the name of the table that you want to output to XML. The > symbol is the output symbol that tells MySQL to output the results to the following filename.

**filename.xml** is the filename that you want to output the XML results.

---

**XML Encryption**

XML encryption was developed to address two common areas not addressed by the Transport Layer Security and Secure Socket Layer protocol (TLS/SSL). TLS/SSL is a very secure and reliable protocol that provides end-to-end security sessions between two parties. XML adds an extra layer of security to TLS/SSL by encrypting part or all of the data being exchanged and by allowing for secure sessions between more than two parties. In other words, each party can maintain secure or insecure sessions with any of the communicating parties, and both secure and non-secure data can be exchanged in the same document. Furthermore, XML encryption can handle both XML and non-XML (e.g. binary) data.

---

**Encrypted Type Element**

EncryptedType is the abstract type from which EncryptedData and EncryptedKey are derived.
that may appear as children of `ds:KeyInfo`.

```
<element ref='ds:KeyInfo' minOccurs='0'/>
```

CipherData is a mandatory element that contains the CipherValue or CipherReference with the encrypted data.

```
<element ref='xenc:CipherData'/>
```

EncryptionProperties can contain additional information concerning the generation of the EncryptedType (e.g., date/time stamp).

```
<element ref='xenc:EncryptionProperties' minOccurs='0'/>
```

Id is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

```
<attribute name='Id' type='ID' use='optional'/>
```

Type is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for mandatory processing in decryption. If the EncryptedData element contains data of Type 'element' or element 'content', and replaces that data in an XML document context, it is strongly recommended the Type attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

```
<attribute name='Type' type='anyURI' use='optional'/>
```

MimeType is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [MIME]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer Encoding may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the MimeType as 'image/png'. This attribute is purely advisory; no validation of the MimeType information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the Type attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text.

```
<attribute name='MimeType' type='string' use='optional'/>
```

EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail. The permitted child elements of the EncryptionMethod are determined by the specific value of the Algorithm attribute URI.

```
<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

CipherData Element

CipherData is a mandatory element that provides the encrypted data. It must either contain the encrypted octet sequence as base64 encoded text of the CipherValue element, or provide a reference to an external location containing the encrypted octet sequence via the CipherReference element.

```
<element name='CipherData' type='xenc:CipherDataType'/>
<complexType name='CipherDataType'>
  <choice>
    <element name='CipherValue' type='base64Binary'/>
    <element ref='xenc:CipherReference'/>
  </choice>
</complexType>
```

CipherReference Element

CipherReference identifies a source which, when processed, yields the encrypted octet sequence CipherReference is used when CipherValue is not supplied directly. The actual value is minOccurs as follows. The CipherReference URI contains an identifier that is dereferenced. Should the CipherReference element contain an OPTIONAL sequence of Transforms, the data resulting from dereferencing the URI is transformed as specified so as to yield the intended cipher value. For example, if the value is base64 encoded within an XML document; the transforms could specify an XPath expression followed by a base64 decoding so as to extract the octets.

```
<element name='CipherReference' type='xenc:CipherReferenceType'/>
<complexType name='CipherReferenceType'>
```
Cipher Reference with Optional Transform feature and Transform Algorithm:

```
<CipherReference URI="http://www.example.com/CipherValues.xml">
  <Transforms>
    <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
      <ds:XPath xmlns:rep="http://www.example.org/repository">
        self::text()[parent::rep:CipherValue[@Id="example1"]]
      </ds:XPath>
    </ds:Transform>
    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
  </Transforms>
</CipherReference>
```

EncryptedData Element

EncryptedData is the core element in the syntax. Not only does its CipherData child contain the encrypted data, but it's also the element that replaces the encrypted element, or serves as the new document root.

```
<element name='EncryptedData' type='xenc:EncryptedDataType'/>
<complexType name='EncryptedDataType'>
  <complexContent>
    <extension base='xenc:EncryptedType'>
    </extension>
  </complexContent>
</complexType>
```

Resources

The information above was obtained from W3C and IBM. For more information, please visit the following links:


---

Learning Objectives

- What is XQL?
- What is an XQL Query?
- Tutorial
- What are the different components of XQL?

Introduction

As more and more information is either stored in XML, exchanged in XML, or presented as XML through various interfaces, the ability to intelligently query our XML data sources becomes increasingly important. XML documents are structured documents – they blur the distinction between data and documents, allowing documents to be treated as data sources, and traditional data sources to be treated as documents.

XQL is a query language designed specifically for XML. In the same sense that SQL is a query language for relational tables and OQL is a query language for objects stored in an object database, XQL is a query language for XML documents. The basic constructs of XQL correspond directly to the basic structures of XML, and XML closely relates to XPath, the common locator syntax used by XSL and XPointers. Since queries, transformation patterns, and links are all based on patterns in structures found in possible XML documents, a common model for the pattern language used in these three applications is both possible and desirable, and a common syntax to express the patterns expressed by that model simplifies the task of the user who must master a variety of XML-related...
The XML Query Language described in this chapter contains several features not found in previously published versions of the language, including joins, links, text containment, and extensible functions. These new features are inspired in large part by discussions stemming from the W3C QL ’98 Workshop, and make it possible to combine information from heterogeneous data sources in powerful ways. Great care has been made to maintain the fundamental simplicity of XQL while adding these features.

This chapter is intended as input for the upcoming W3C Query Language Activity, and for the further development of XPath.

XML Query Language

Traditionally, structured queries have been used primarily for relational or object-oriented databases, and documents were queried with relatively unstructured full-text queries. Although quite sophisticated query engines for structured documents have existed for some time, they have not been a mainstream application.

In the last year, a number of very different approaches to querying XML have been proposed, with several distinct perspectives on what constitutes a query. Several particularly interesting proposals have come from the semi-structured database community, including XML-QL and Lorel, and adopt semi-structured approaches to XML. This proposal incorporates several ideas from those languages into XQL.

XQL was designed to be used in a number of different XML environments, using a syntax that may be used in XML attributes, embedded in programming languages, or incorporated in URIs. From the beginning, we have endeavored to keep the language simple and small, and we have been careful not to add functionality that would make it difficult to implement XQL. During the last year, we have been persuaded to add several powerful new features that allow users to combine information from multiple sources, use the relationships expressed in links as part of a query, and search based on text containment. Queries that can make use of information in multiple documents allow the information contained in those documents to be reused in ways not foreseen by the people who created the original documents. This is extremely useful when many documents or data sources may each contain part of the information needed on a given topic. For instance, suppose one document contains a set of recommended books for a given course of study, another lists books and prices for a store, and third contains a set of reviews of books. A query can be constructed to list recommended books, their prices, and the reviews they have received.

XQL is closely related to XPath, and we hope to be able to maintain compatibility with XPath as it evolves. We see XQL as complementary to XSLT, which may be used for sophisticated reshaping and formatting of query results.

XML as a Data Model

An important motivation for the design of XQL is the realization that XML has its own implied data model, which is neither that of traditional relational databases nor that of object-oriented or object-relational databases. In XQL, a document is an ordered, labelled tree, with nodes to represent the document entity, elements, attributes, processing instructions, and comments. The model is compatible with the XML Information Set (http://www.w3.org/XML/Group/1999/04/WD-xml-infoset-19990428.html).

It is important to note that the relationships among data contain a large proportion of the information contained in a document, which is one of the reasons that structured document formats like XML are useful in the first place. The original formulation of XQL was based completely on the tree structure of XML documents:

1. Hierarchy
   1. parent/child
   2. ancestor/descendant

2. Sequence (within a sibling list or in document order)

3. Position (within a sibling list or in document order)
   1. absolute
   2. relative
   3. ranges

These relationships have long been basic to the XPointer model, and are now reflected in XPath in the form of axes. In XQL, all queries use the child axis, so we will speak in terms of parent/child and ancestor/descendant relationships rather than use the term Locator Path from the XPath Working Draft.

The current draft extends this model to support the following:

1. Ad-hoc relationships established via joins
2. Dereferencing of links

Joins allow subtrees of documents to be combined in queries; links allow queries to support references as well as tree structure.

What is an XML Query?

In XQL, a query returns XML document nodes from one or more XML documents. To examine the characteristics of an XQL query, it is useful to consider four basic questions about the environment in which a query takes place:

1. What is a database?
2. What is the query language?
3. What is the input to a query?
4. What is the result of a query?

The following table provides a brief answer to each of these questions, including a comparison with the SQL query language, which is widely used for querying relational databases:

<table>
<thead>
<tr>
<th>SQL</th>
<th>XQL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Queries are done in SQL, a query language that uses the structure of tables as a basic model.

The FROM clause determines the tables which are examined by the query.

The result of a query is a table containing a set of rows; this table may serve as the basis for further queries.

Queries are done in XQL, a query language that uses the structure of XML documents as a basic model.

A query is given a list of input nodes from one or more documents.

The result of a query is a list of XML document nodes, which may serve as the basis for further queries.

From the preceding table, it should be clear that document nodes play a central role in XQL queries. These nodes are an abstraction. Any real XQL implementation will find some concrete way to implement the nodes used in queries. For instance, XQL engines may represent the input to a query via DOM nodes, XSL nodes, index structures, or XML text. Any of these might also be used to represent the results of queries; in addition, hyperlinks or other references into the original document might be used, a new virtual document might be created, or DOM Level Two TreeWalkers or Iterators might be used.

The nodes which form the input to a query may come from a variety of different sources. They may be the result of a prior query, the contents of a document repository, the nodes from a Document Object Model Nodelist, or any other source that identifies nodes from one or more documents. XQL does not specify how these nodes are brought to the query. Current XQL implementations take a variety of approaches, including the following: using Document Object Model subtrees as the basis for a query, querying whole documents supplied as the input to a Unix-style pipe, reading a document from the command line, using data dictionaries or repository directory structures to identify nodes to be queried, and identifying documents using a URL. This proposal adds support for merging information from heterogeneous data sources using joins.

In XQL, nodes have identity, and they retain their identity, containment relationships, and sequence in query results. Grouping operators allow levels of a tree to be omitted, while still retaining the relative sequence and containment of the nodes which are returned by a query. Joins allow subtrees from one data source to be inserted into another document subtree, subject to the join conditions. Link functions are similar to joins, allowing a hypertext link in a document to be replaced by the node or nodes to which it refers. Some functions in XQL return values, which may be boolean, integer, or string. These values are also treated as nodes in the query model.

### XQL Tutorial

Before going into further detail, we feel it would be helpful to present some typical XQL queries to help convey a feeling for the language. This tutorial discusses the simplest XQL queries, which are also likely to be the most common. In this tutorial, we will present a quick overview of XQL without taking the time to be precise.

A simple string is interpreted as an element name. For instance, this query specification returns all `<table>` elements:

```
table
```

The child operator (`"/"`) indicates hierarchy. This query specification returns `<author>` elements that are children of `<front>` elements:

```
front/author
```

The root of a document may be indicated by a leading `/` operator:

```
/novel/front/author
```

**Ed. Note:** In XQL, the root of a document refers to the document entity, in the technical XML sense, which is basically equivalent to the document itself. It is not the same as the root element, which is the element that contains the rest of the elements in the document. The document root always contains the root element, but it may also contain a doctype, processing instructions, and comments. In this example, `<novel>` would be the root element.

Paths are always described from the top down, and unless otherwise specified, the right-most element on the path is returned. For instance, in the above example, `<author>` elements would be returned.

The content of an element or the value of an attribute may be specified using the equals operator (`"="`). The following returns all authors with the name "Theodore Seuss Geisel" that are children of the `<front>` element:

```
front/author='Theodore Seuss Geisel'
```

Attribute names begin with `"@"`. They are treated as children of the elements to which they belong:

```
front/author/address/@type='email'
```

The descendant operator (`"//"`) indicates any number of intervening levels. The following shows addresses anywhere within `<front>`:

```
front/address
```

When the descendant operator is found at the start of a path, it means all nodes descended from the document. This query will find any address in the document:

```
//address
```

The filter operator (`"[ ]"`) filters the set of nodes to its left based on the conditions inside the brackets. The following query returns addresses; each of these addresses must have a `"type"` attribute with the value "email":

```
front/author/address[@type='email']
```

Note that `"address[@type='email']"` returns addresses, but `"address/@type='email'"` returns type attributes.

Multiple conditions may be combined using Boolean operators.

```
front/author='Theodore Seuss Geisel'[@gender='male' and @shoesize='9EEEE']
```

Brackets are also used for subscripts, which indicate position within a document. The following refers to sections 1, 3, 4, 5, and 8, plus the last section:

```
section[1,3 to 5, 8, -1]
```
sections whose level attributes have the value "3"; in other words, it returns the first three "level3" sections:

```
section[level='3'][1 to 2]
```

Now that we know the basics, let's take a look at a document and try some XQL queries on it. The following is an invoice document. Traditionally, invoices are often stored in databases, but invoices are both documents and data. XQL is designed to work on both documents and data, provided they are represented via XML through some interface. This document will be the basis for the sample queries that follow:

```
<?xml version="1.0"?>
<invoicecollection>
  <invoice>
    <customer>Wile E. Coyote, Death Valley, CA</customer>
    <annotation>
      Customer asked that we guarantee return rights if these items should fail in desert conditions.
      This was approved by Marty Melliore, general manager.
    </annotation>
    <entries n="2">
      <entry quantity="2" total_price="134.00">
        <product maker="ACME" prod_name="screwdriver" price="80.00"/>
      </entry>
      <entry quantity="1" total_price="20.00">
        <product maker="ACME" prod_name="power wrench" price="20.00"/>
      </entry>
    </entries>
  </invoice>
  <invoice>
    <customer>Camp Mertz</customer>
    <entries n="2">
      <entry quantity="2" total_price="32.00">
        <product maker="BSA" prod_name="left-handed smoke shifter" price="16.00"/>
      </entry>
      <entry quantity="1" total_price="13.00">
        <product maker="BSA" prod_name="snipe call" price="13.00"/>
      </entry>
    </entries>
  </invoice>
</invoicecollection>
```

Now let's look at some sample queries. For these examples, we will present query results as text, using a serialization approach described in the section "Query Results and Serialization". In general, XQL queries return lists of nodes, which may be represented in any way convenient to the environment in which the query is performed, e.g. as DOM nodes, serialized XML text, XPointers, hyperlinks, or by creating an iterator to navigate the results. Since XML text is easily read, we find it suitable as a way of representing results in our examples.

Suppose we wanted to see just the customers from the database. We could do the following query:

**Query:**
//customer

**Result:**
```
<xql:result>
  <customer>Wile E. Coyote, Death Valley, CA</customer>
  <customer>Camp Mertz</customer>
</xql:result>
```

We might want to look at all the products manufactured by BSA. This query would do the trick:

**Query:**
//product[@maker='BSA']

**Result:**
```
<xql:result>
  <product maker="BSA" prod_name="left-handed smoke shifter" price="16.00"/>
  <product maker="BSA" prod_name="snipe call" price="13.00"/>
</xql:result>
```

Filters are particularly useful when specifying conditions on paths that are not the same as what is returned. For instance, the following query returns the products ordered by Camp Mertz:

**Query:**
//invoice[@customer='Wile E. Coyote, Death Valley, CA']/product

**Result:**
```
<xql:result>
  <product maker="ACME" prod_name="screwdriver" price="80.00"/>
  <product maker="ACME" prod_name="power wrench" price="20.00"/>
</xql:result>
```

This is the end of the tutorial, which covers only the most basic features of XQL. For examples illustrating newer or more advanced features, such as return operators, sequence, joins, references, and user-defined functions, see the appropriate parts of the next section.

**XQLExpressions**
An XQL query is always evaluated for a context, which is a list of document nodes. The initial context for a query is known as the start context. In XQL, the nodes in a start context may come from different documents, and even if they are in the same document, there is no assumption that they come from contiguous portions of the document. Some XQL operators establish a new context in which a subexpression will be evaluated; for instance, in the expression "author/name", "author" is evaluated in the start context. For each author, the "/" operator establishes a new context consisting of the children of that author, and "name" is evaluated in that context. The operators that establish a new context are "/", "/", and "."

**Ed. Note:** In XSL, expressions are evaluated with respect to a node which is called the context node. Our use of the term "context" is intended to allow semantic consistency with XSL Patterns without imposing unnecessary restrictions on the query language. As a consequence, XSL Patterns are defined in terms of children of the context node, and XQL queries are defined in terms of the context node directly. We maintain the correspondence of XSL Pattern definitions and XQL definitions by constructing an imaginary context node that contains the nodes of the context, and allowing the XSL term "." to map to this context node.

**Terms**

The following expressions are terms, which select particular nodes from the context based on the type or name of the node:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>element name</td>
</tr>
<tr>
<td>*</td>
<td>element name with wildcards</td>
</tr>
<tr>
<td>@n</td>
<td>attribute name</td>
</tr>
<tr>
<td>@*</td>
<td>attribute name with wildcards</td>
</tr>
<tr>
<td>text()</td>
<td>text node</td>
</tr>
<tr>
<td>*</td>
<td>text node</td>
</tr>
<tr>
<td>comment()</td>
<td>comment node</td>
</tr>
<tr>
<td>pi()</td>
<td>processing instruction</td>
</tr>
<tr>
<td>pi(&quot;v&quot;)</td>
<td>processing instruction with target</td>
</tr>
<tr>
<td>:</td>
<td>context node</td>
</tr>
</tbody>
</table>

**Namespaces and names**

In XML expressions, names may be associated with namespace prefixes. A namespace prefix can be declared using a variable declaration. In the following query, the first line declares "b" to be a variable equivalent to the namespace URL "http://www.TwiceSoldTales.com". The second line of the query searches for all <book> elements belonging to this namespace:

```xml
$b := "http://www.TwiceSoldTales.com";
//b:book
```

An XML document may well use a different namespace prefix for the same namespace URI. Matching is done on the basis of the namespace URI, not the prefix associated with it in the document or in the XQL query.

XQL expressions can explicitly state whether namespaces should be taken into account when matching node names:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Any element named &lt;table&gt;, regardless of the namespace to which it belongs.</td>
</tr>
<tr>
<td>html:table</td>
<td>Any element named &lt;table&gt; that belongs to the namespace indicated by the prefix &quot;html&quot;</td>
</tr>
<tr>
<td>*</td>
<td>Any element, regardless of the namespace to which it belongs.</td>
</tr>
<tr>
<td>table</td>
<td>Any element named &lt;table&gt; for which no namespace has been declared.</td>
</tr>
<tr>
<td>html:*</td>
<td>Any element belonging to the namespace associated with the prefix &quot;html&quot;.</td>
</tr>
<tr>
<td>*</td>
<td>Any element for which no namespace has been declared.</td>
</tr>
<tr>
<td>html:*</td>
<td>Any element for which a namespace has been declared.</td>
</tr>
</tbody>
</table>

The same conventions apply to attribute names. In attribute names, the attribute prefix comes before the namespace prefix:

```xml
@lib:isbn
```

Namespaces are preserved in the output of a query. To change the namespaces of nodes in the output, use the Renaming Operator.

**Comparisons**

Comparisons add constraints based on the content or value of nodes. Consider the following examples:

```xml
author = "Washington Irving"
@id = "id-sec-0203"

text() = "When that Aprille with his shoures soughte"
```

Regardless of the node type on the left hand of the comparison, it is compared to the value on the right. For systems that use a schema that supports data types, they are used in comparisons:

```xml
books[pub_date < date("1990-01-01")]
books[pub_date lt date("1990-01-01")]
```

Since some environments in which XQL is used have restricted character sets, e.g. URIs or queries stored in attribute values, many comparisons have an alternative syntax that meets the syntactic constraints of these environments. For instance, the following two queries are equivalent:

```xml
books[pub_date < date("1990-01-01")]
books[pub_date lt date("1990-01-01")]
```
The following comparison operators are available in XQL:

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td><code>n=&quot;value&quot;</code></td>
</tr>
<tr>
<td>Case insensitive comparison</td>
<td><code>n ieq &quot;value&quot;</code></td>
</tr>
<tr>
<td>Inequality</td>
<td><code>n !=&quot;value&quot;</code></td>
</tr>
<tr>
<td>Text containment</td>
<td><code>n contains &quot;value&quot;</code></td>
</tr>
<tr>
<td>Case insensitive text containment</td>
<td><code>n icontains &quot;value&quot;</code></td>
</tr>
</tbody>
</table>

Text comparisons support the wildcard characters "*" and "?". Consider the following example:

Data:

```xml
<editor>
    <name>
        <first> Ramesh </first>
        <last> Lekshmynarayanan </last>
    </name>
</editor></customer>
```

Query:

```xml
//(<editor[contains="Leksh*" ])
```

The value "Leksh*" matches the name "Lekshmynarayanan", and the `<editor>` element is returned.

The following operators may be defined in XQL environments that support data types:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than</td>
<td><code>n &lt; value</code></td>
</tr>
<tr>
<td>Less than or equals</td>
<td><code>n &lt;=value</code></td>
</tr>
<tr>
<td>Greater than</td>
<td><code>n &gt; value</code></td>
</tr>
<tr>
<td>Greater than or equals</td>
<td><code>n &gt;=value</code></td>
</tr>
</tbody>
</table>

### Hierarchy and Filters

These operators establish a new search context and evaluate a subexpression within that context. In this table, Q1 and Q2 are used to denote arbitrary XQL expressions.

<table>
<thead>
<tr>
<th>Q1/Q2</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent/child</td>
<td>Children of nodes that satisfy Q1, evaluated in the current context, such that the children satisfy Q2. Q2 is evaluated separately for the child list of each node in Q1; the nodes to which each child list evaluates are unioned together.</td>
</tr>
<tr>
<td>ancestor/descendant</td>
<td>Descendants of nodes that satisfy Q1, evaluated in the current context, such that the descendants satisfy Q2. Q2 is evaluated separately for each child list of each node in Q1, and recursively for each node in the child list; the nodes to which each child list evaluates are unioned together.</td>
</tr>
<tr>
<td>filter</td>
<td>Nodes that satisfy Q1, evaluated in the current context, containing children that satisfy Q2. Q2 is evaluated separately for the child list of each node in Q1; the nodes to which each child list evaluates are unioned together.</td>
</tr>
<tr>
<td>Q1[poslist]subscript</td>
<td>Nodes that satisfy Q1, evaluated in the current context, whose position in the evaluation list is contained in the poslist.</td>
</tr>
</tbody>
</table>

### Boolean and Set Operators

Terms or other XQL expressions may be combined using boolean operators and set operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td><code>not(q)</code></td>
<td>All nodes in the context for which the expression q evaluates to null.</td>
</tr>
<tr>
<td>union</td>
<td><code>q1 union q2</code></td>
<td>The union of q1 and q2, evaluated in the context.</td>
</tr>
<tr>
<td>intersection</td>
<td><code>q1 intersect q2</code></td>
<td>The intersection of q1 and q2, evaluated in the context.</td>
</tr>
<tr>
<td>union</td>
<td>`q1</td>
<td>q2`</td>
</tr>
<tr>
<td>both</td>
<td><code>q1 – q2</code></td>
<td>If both q1 and q2 are non-empty, returns q1 union q2; if either is empty, returns the empty list.</td>
</tr>
<tr>
<td>or</td>
<td><code>q1 or q2</code></td>
<td>(Boolean) If the union of q1 and q2, evaluated in the context, is non-empty, returns true; else, returns false.</td>
</tr>
<tr>
<td>and</td>
<td><code>q1 and q2</code></td>
<td>(Boolean) If the intersection of q1 and q2, evaluated in the context, is non-empty, returns true; else, returns false.</td>
</tr>
</tbody>
</table>
The "both" operator was introduced because we found that many queries use filters to express constraints on the same data that is returned outside the filter, resulting in expressions that are rather redundant. For instance, the following query uses filters to express that only invoices for the customer named "Wile E. Coyote" that also contain products are of interest, and both the customer name and the set of products should be returned:

```xml
//*[customer[name='Wile E. Coyote'] and ./product]/(customer | ./product)
```

Using the "both" operator, this same query can be expressed more concisely:

```xml
//*[customer[name='Wile E. Coyote'] ~ ./product]
```

Note that the "both" operator is neither the boolean "or" operator nor the set intersection operator. The expression "customer intersect product" always returns an empty result since no element is ever simultaneously a `<customer>` element and a `<product>` element. The "both" operator is used to specify conditions which must simultaneously be satisfied for the context.

**Grouping Operator**

It is often useful to group results using the structure of the original document. For instance, a query that lists the products on invoices might want to group products by invoice, placing each group of products within an invoice tag. XQL provides a grouping operator that provides exactly this functionality. In the following query, the element to the left of the curly braces (the Grouping Element) is used to group the results of the query within the braces:

```xml
//invoice { .//product }
```

For each grouping element matched by the query, the grouping operator creates an empty element with the same name. The results of the query contained within the curly braces are then appended to this new node as children. If we apply this query to the invoice data presented in the tutorial, we obtain these results:

```xml
<xql:result>
  <invoice>
    <product maker="ACME" prod_name="screwdriver" price="80.00"/>
    <product maker="ACME" prod_name="power wrench" price="20.00"/>
  </invoice>
  <invoice>
    <product maker="BSA" prod_name="left-handed smoke shifter" price="16.00"/>
    <product maker="BSA" prod_name="snipe call" price="13.00"/>
  </invoice>
</xql:result>
```

Complex queries that use the grouping operator can be made more readable by the appropriate use of whitespace, eg:

```xml
invoice { .[name contains 'Coyote'] {
  name | address
} ~ entries {
  ./product[@maker='ACME']
}}
```

**Sequence**

XQL defines the following operators for sequence:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>before a after b</td>
<td>Returns a list of all &quot;a&quot;s that precede a &quot;b&quot;.</td>
</tr>
<tr>
<td>after a after b</td>
<td>Returns a list of all &quot;a&quot;s that occur after a &quot;b&quot;.</td>
</tr>
<tr>
<td>list concatenation a, b</td>
<td>Returns a list containing all &quot;a&quot;s, followed by all &quot;b&quot;s. Useful for specifying order in return lists.</td>
</tr>
</tbody>
</table>

The list concatenation operator is used to specify order in return lists. In general, XQL operators maintain document order; the concatenation operator allows an order to be specified within a return list. For instance, the following query specifies that the order of the returned results should be author, then title, then isbn:

```xml
//book //author, title, isbn
```

If there is more than one author, all authors will be listed before the title.

In systems where XML is used mainly to represent data from object oriented systems or relational databases, sequence may not be particularly important. However, sequence is important in documents, and it also can be useful in data-oriented applications where the markup does not clearly indicate the role of each element. Consider the following table, which lists the latest scores for some fictitious sport:

<table>
<thead>
<tr>
<th>Western League</th>
<th>Southern League</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aardvarks 12</td>
<td>Tortoises 25</td>
</tr>
<tr>
<td>Weasels 10</td>
<td>Hares 0</td>
</tr>
<tr>
<td>Mosquitos 17</td>
<td>Platypii 17</td>
</tr>
<tr>
<td>Slugs 2</td>
<td>Amoebae 16</td>
</tr>
</tbody>
</table>

The markup for this table looks like this:

```html
<table width="50%" border="2">
  <thead>
    <tr>
      <th colSpan="2">emph</th>
    </tr>
    <tr>
      <td width="50%">emph</td>
      <td width="50%">emph</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td width="50%">emph</td>
      <td width="50%">emph</td>
    </tr>
    <tr>
      <td width="50%">emph</td>
      <td width="50%">emph</td>
    </tr>
  </tbody>
</table>
```
Purists may object that this is not particularly good markup, since it does not clearly distinguish the leagues from the scores. We agree, and when we write our own documents, we would write them differently; however, there is a lot of mediocre markup in the real world, and when querying documents, we do not have the luxury of rewriting them first. Therefore, we feel that a query language should be able to manage data like that shown above.

To find all the latest scores for the Western League, we can use the following query:

```
table//((tr after (tr contains "Western League")) before (tr contains "Southern League"))
```

**Ed. Note:** Sequence is handled by axes in XPath. We believe that an XML query language should provide some means for allowing sequence in queries, and that various approaches should be considered. The approach discussed here has advantages in expressing relationships among multiple nodes, especially when comparisons are to be made only within the descendants of a particular node.

## Functions

Most of the functions of XQL have been taken directly from XSL Pattern Language. A few functions have been added, many more have been omitted because we found them to be less relevant in a pure query environment than in a general purpose transformation environment.

### Collection functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute()</td>
<td>Returns the attributes in the context. If a name argument is supplied, returns the attribute with the given name.</td>
</tr>
<tr>
<td>comment()</td>
<td>Returns the comments in the context.</td>
</tr>
<tr>
<td>element()</td>
<td>Returns the elements in the context. If a name argument is supplied, returns the elements with the given name.</td>
</tr>
<tr>
<td>entity-ref()</td>
<td>Returns the entity references in the context. XQL operates on a view of the document in which all entity references are expanded; this function is the only way to locate entity references in XQL.</td>
</tr>
<tr>
<td>node()</td>
<td>Returns all nodes in the context.</td>
</tr>
<tr>
<td>pi()</td>
<td>Returns the processing instructions in the context. If a target argument is supplied, returns the processing instructions with the given target.</td>
</tr>
<tr>
<td>text()</td>
<td>Returns the text nodes in the context. For the sake of text nodes, XQL assumes that CDATA sections are treated as text, adjacent text nodes are merged, and entity references are expanded.</td>
</tr>
</tbody>
</table>

### Extensible Functions

Many XQL implementations are part of a programming environment. In these environments, it is helpful to allow users to write their own functions, which may be used in queries. This must be done in a language-independent manner, since XQL implementations have been done in a variety of languages, including C++, Java, Haskell, and Perl. To allow user-defined functions to be written, XQL provides a function called "function()".

Suppose a user wanted to add a function that computes the average for a list of values. The user could write a function called "average" and call it in an XQL query like this:

```
average(property//price)
```

User-defined functions are typically written in the language environment of the XQL implementation; for instance, if the XQL implementation is written in Java, user-defined functions are generally written as Java functions. All XQL functions are passed the list of nodes in the current context. If the function has parameters, these are passed as strings to the XQL function. Typically, the function will evaluate these parameters as queries against the current context; for instance, the user code that implements the "average" function might first execute the query "property//price" for the current context to obtain a set of <price> elements, then compute the average of these elements.
The result of a function call is also a nodelist. If a single value is to be returned, such as a string or a number, it should be returned as an element node of that type:

\[ <xql:number> 112,000.47 </xql:number> \]

The available set of types that may be returned by functions is described in the section "Query Results and Serialization", which follows the current section. If a function is called with the wrong parameters, this may be communicated by returning an \(<xql:warning>\) element in the result:

\[ <xql:warning> "average" requires numeric values for the nodes to be averaged </xql:warning> \]

Ed. Note: Some vendors have asked that extensible operators be provided as well. This would be a useful feature; so far, we have not found a clean design for extensible operators in XQL.

Issue (function-namespace): There are differing opinions as to whether namespaces add significant value as vendors and users add functions to XQL.

References

Ed. Note: The ideas in this section are exploratory, and have not yet been incorporated into XQL.

There is currently no syntax for dereferencing links in XQL, but this is clearly needed in many applications. XSL provides the "id()" function, which returns the element containing a given id. For instance, the following would evaluate to the node pointed to by an HREF attribute in an \(<A>\) element:

\[ A/id(@HREF) \]

From an XQL perspective, this is actually a kind of join. However, the above syntax is less complex than the equivalent join syntax:

\[ A/id[@HREF]//*[@id=$h] \]

We need functionality similar to id(), extending this functionality to incorporate any kind of link, not just ID/IDREF. Let's create a function called \( \text{ref}() \) which returns the node or nodes to which an XPointer or HTML HREF points.

\[ A/ref(@HREF) \]

One advantage of the join syntax is that it allows the type of the referenced node to be specified. It may be useful to be able to specify this as a further parameter to the function. Let's allow the type of the referenced node to be specified as a second parameter to the function. For instance, the following will return the referenced node only if it is a 'table' element; otherwise, it will return null:

\[ A/ref(@HREF, "table") \]

It may also be helpful to specify further parameters, e.g. to limit the scope of the reference to the current document, the local repository, or some other identifiable scope.

It is frequently useful to be able to identify the references to a particular node from other nodes. For instance, if we are thinking of deleting something from a document, we may want to know if it is referenced. For this purpose, it may be useful to introduce another function that returns all nodes that reference a particular node. If we call this function "backref()", it might look like this:

\[ A/backref(table[0]) \]

Issue (ref-scope): Backwards references will also need to be scoped somehow, and not all systems will want to support them, due to implementation overhead.

References can also be used to specify the URLs of documents used in queries:

\[ ref("http://www.amazon.com")//book[@title contains "Alhambra"] \]

Joins

Ed. Note: Joins are a new feature in XQL. The approach to joins discussed in this section comes largely from Peter Fankhauser of the GMD-IPSI and Harald Schöning of Software AG. Gerald Huck of the GMD-IPSI has been particularly helpful in refining the initial model. There is some preliminary implementation experience with this approach.

In many environments, it is useful to be able to combine information from multiple sources to create one unified view. For instance, suppose we have a source of books and a source of reviews:

```xml
<Book>
  <isbn>84-7169-020-9</isbn>
  <title>Tales of the Alhambra</title>
  <author>Washington Irving</author>
</Book>

<Review>
  <isbn>84-7169-020-9</isbn>
  <title>Tales of the Alhambra</title>
  <reviewer>Ricardo Sanchez</reviewer>
  <comments>
    A romantic and humorous account of the time that the author of "The Legend of Sleepy Hollow" lived in an Arabian palace in Spain.
  </comments>
</Review>
```

We may want to combine these to create a view of the book that includes the comments found in reviews:

```xml
<Book>
  <isbn>84-7169-020-9</isbn>
  <title>Tales of the Alhambra</title>
  <author>Washington Irving</author>
</Book>
```
This amounts to inserting information from the review into the book. If we had a database that consisted only of this one book and this one review, we could obtain the desired result with this query:

```
/book {
  isbn | title | author | //review { reviewer | comments }
}
```

If we are using a database with many books and many reviews, the above query would include the whole list of reviews in every single book, not just the reviews for the book in question. We need some way to restrict our reviews to those that have the same ISBN number as the book. We will do this by introducing correlation variables. In the following example, "$i := isbn" assigns the variable "$i" to the evaluation of isbn in the context of each book. The expression "/review[isbn=$i]" restricts the reviews to those that match "$i".

```
/book[$i:=isbn] {
  $i | title | author | //review[isbn=$i] { reviewer | comments }
}
```

**Ed. Note:** Although filters and variable bindings both use square bracket notation, variable bindings do not filter results. For instance, the expressions "/book" and "/book[$i:=isbn]" will always return the same set of books, whether or not any <isbn> elements are present.

Variable bindings propagate as new search contexts are created; when a new context is created, e.g. as the result of a child or descendant operator, it inherits all variable bindings that are active. This allows bindings declared high in the document hierarchy to be used for joins performed lower down.

If a correlation variable is bound to a subexpression that evaluates to more than one result, any value in the list of results will be used as the basis for a join. To be precise, "list1 relop list2" evaluates to "all e1 in list1 such that for some e2 in list2, e1 relop e2 is satisfied".

The following query returns books whether or not they have an isbn; reviews are returned only if they have a matching isbn:

```
/book[$i:=isbn] {
  $i | title | author
  | //review[isbn=$i] { reviewer | comments }
}
```

**Ed. Note:** In this example, it seems intuitive to say that you can't join on null - a book with no isbn does not match all reviews that have no isbn. On the XML mailing list, there is some difference of opinion as to whether it should be possible to join on null.

In XQL, square brackets are used for three distinct things that can not be mixed: subscripts, filters, and variable bindings. If you want both a filter and a variable binding, you must use separate sets of brackets:

```
/book[isbn][{$i:=isbn}] {
  $i | title | author
  | //review[isbn=$i] { reviewer | comments }
}
```

**RenamingOperator**

The nodes in a list may be renamed using the renaming operator "->". In joins, this can be used to reflect a meaningful name that describes the synthesized result:

```
/book[isbn][{$i:=isbn}] -> BookWithReviews {
  $i | title | author
  | //review[isbn=$i] { reviewer | comments }
}
```

The renaming operator may also be used to adjust namespaces in query results. Since renaming changes the name of a node, it also changes the namespace. For instance, suppose <book> is in the namespace of "http://www.TwiceSoldTales.com", and we rename the <book> element to <livre>:

```
/livre
```

We can assume that <livre> is not defined in the namespace associated with "http://www.TwiceSoldTales.com". Since renaming often creates element names that do not exist in the original namespace, renaming in XQL does not keep the namespace of the original node name. This property of the renaming operator can be used to remove namespaces; for instance, the following query places <book> elements in the default namespace, regardless of their original namespace:

```
/book->book
```

New namespace prefixes may be explicitly applied with the rename operator:

```
/book := book
```
Precedence of Operators

XQL expressions are evaluated from left to right. The following table shows the precedence of operators in XQL:

<table>
<thead>
<tr>
<th>Query Operators by Decreasing Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grouping</td>
</tr>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>Renaming</td>
</tr>
<tr>
<td>Grouping</td>
</tr>
<tr>
<td>Path</td>
</tr>
<tr>
<td>Comparison, Assignment</td>
</tr>
<tr>
<td>Intersection</td>
</tr>
<tr>
<td>Union</td>
</tr>
<tr>
<td>Negation</td>
</tr>
<tr>
<td>Conjunction</td>
</tr>
<tr>
<td>Disjunction</td>
</tr>
<tr>
<td>Sequence</td>
</tr>
<tr>
<td>End of Statement</td>
</tr>
</tbody>
</table>

Parentheses may be used for grouping:

```xml
<author | editor/name
```

Query Results and Serialization

In some environments, the results of a query are returned as XML text. XQL defines a serialization format to allow the results of queries to be returned as well-formed XML documents. Namespaces are used to distinguish tags belonging to the serialization format from tags returned by the query. When query results are serialized, they are wrapped in an `<xql:result>` element:

```xml
<xql:result xmlns:xql="http://www.metalab.unc/xql/serialization">
  <customer>Wile E. Coyote, Death Valley, CA</customer>
  <customer>Camp Mertz</customer>
</xql:result>
```

The reason for this is that a well-formed XML document may have only one root element, and queries may return any number of results. Other XQL serialization elements are used to return values from functions, provide additional information about a query, or indicate errors or warnings. The following elements are defined in the XQL serialization namespace:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xql:result&gt;</code></td>
<td>Surrounds the serialized results of the query.</td>
</tr>
<tr>
<td><code>&lt;xql:query&gt;</code></td>
<td>Optional. Contains the original query string. Useful for debugging.</td>
</tr>
<tr>
<td><code>&lt;xql:true&gt;</code></td>
<td>Returned by boolean functions.</td>
</tr>
<tr>
<td><code>&lt;xql:false&gt;</code></td>
<td>Returned by boolean functions.</td>
</tr>
<tr>
<td><code>&lt;xql:number&gt;</code></td>
<td>Returned by numeric functions.</td>
</tr>
<tr>
<td><code>&lt;xql:text&gt;</code></td>
<td>Returned by text functions.</td>
</tr>
<tr>
<td><code>&lt;xql:attribute name=&quot;attributeName&quot; value=&quot;attributeValue&quot;&gt;</code></td>
<td>Used to return attributes when they are returned outside of the attribute list of an element.</td>
</tr>
<tr>
<td><code>&lt;xql:declaration&gt;</code></td>
<td>Used to return the XML declaration when it is returned in a query.</td>
</tr>
<tr>
<td><code>&lt;xql:error&gt;</code></td>
<td>Used to indicate an error in the query. The content of this element explains the error.</td>
</tr>
<tr>
<td><code>&lt;xql:warning&gt;</code></td>
<td>Used to indicate a warning. The content of this element explains the warning.</td>
</tr>
</tbody>
</table>

1. Definition of XQuery
XQuery is a query language under development by the World Wide Web Consortium (W3C) and makes it possible to efficiently and easily extract information from native XML databases and relational databases that store XML data.

Every query consists of an introduction and a body. The introduction establishes the compile-time environment such as schema and module imports, namespace and function declarations, and user-defined functions. The body generates the value of the entire query. The structure of XQuery shows in Figure 1.

Figure 1. Structure of XQuery

```
Comment: (: Sample version 1.0 :)  

Namespace Declaration: declare namespace my = "urn:foo";

Function Declaration: declare function my:fact($n) {
    if($n < 2)
        then 1
    else $n * my:fact($n - 1)
};

Global Variable: declare variable $my:ten {my:fact(10)};
```

```
Constructed XML: <table>
    for $i in 1 to 10
        return
        <tr>
            <td>10!/{$i}! = {$my:ten div my:fact($i)}
        </tr>
    </table>
```

2. XQuery versus Other Query Languages

2.1 XQuery versus XPath and XSLT

XQuery, XPath, XSLT, and SQL are good query languages. Each of these languages has its own advantages in diverse situations, so XQuery cannot substitute for them at every task. XQuery is built on XPath expressions. XQuery 1.0 and XPath 2.0 shares the same data model, the same functions, and the same syntax. Table 1 shows the advantages and the drawbacks of each query language.

Table 1. XQuery versus XPath and XSLT

<table>
<thead>
<tr>
<th></th>
<th>Advantage</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>XQuery</td>
<td>1. expressing joins and sorts</td>
<td>1. XQuery implementations are less mature</td>
</tr>
<tr>
<td></td>
<td>2. manipulating sequences of values and nodes</td>
<td>than XSLT ones</td>
</tr>
<tr>
<td></td>
<td>in arbitrary order</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. easy to write user-defined functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>including recursive ones</td>
<td></td>
</tr>
</tbody>
</table>
4. allows users to construct temporary XML results in the middle of a query, and then navigate into that XPath 1.0

1. convenient syntax for addressing parts of an XML document
2. selecting a node out of an existing XML document or database

1. cannot create new XML
2. cannot select only part of an XML node
3. cannot introduce variables or namespace bindings
4. cannot work with date values, calculate the maximum of a set of numbers, or sort a list of strings

XSLT 1.0

1. recursively processing an XML document or translating XML into HTML and text
2. creating new XML or part of existing nodes
3. introducing variables and namespaces

1. cannot be addressed without effectively creating a language like XQuery
2. cannot work with sequences of values

2.2 XQuery versus SQL

XQuery has similarities to SQL in both style and syntax. The main difference between XQuery and SQL is that SQL focuses on unordered sets of “flat” rows, while XQuery focuses on ordered sequences of values and hierarchical nodes.

3. XQuery Expressions

3.1 FLWOR expressions

FLWOR expressions are important part of XQuery. FLWOR is pronounced "flower". This name comes from the FOR, LET, WHERE, ORDER BY, and RETURN clauses that organize the expressions. The FOR and LET clauses can come out any number of times in any order. The WHERE and ORDER BY clauses are optional. However, these clauses must be shown in the order given if they are used. The RETURN clause should exist.

XQuery permits you to use join queries in a similar way to SQL. This example is depicted in Example 1 as a join between the videos table and the actors table.

Example 1.

```xml
let $doc := .
for $v in $doc//video,
  $a in $doc//actors/actor
where ends-with($a, 'Lisa')
  and $v/actorRef = $a/@id
order by $v/year
return $v/title
```

The LET clause states a variable assignment. In this case, the query initializes it to doc (‘videos.xml’), or a query’s result places a document in a database. The FOR clause describes a mechanism for iteration: one variable processes all the videos in turn, another variable processes all the actors in turn. In this case, the query processes the pairs of videos and actors. The WHERE clause selects tables in which you are interested. In this case, you want to know that the actor shows in video table with the name ending with “Lisa”. The ORDER BY clause obtains the results in sorted order. In this case, you desire to have a result with the videos in order of their release date. The RETURN clause at the end of an expression informs the system what information you want to get back. In this case, you want the video’s title.

3.2 Conditional expression

XQuery offers IF, THEN, and ELSE clause, conditional expression. The ELSE clause is obligatory. The reason is that each expression in XQuery should return a value. A query is showed at example 2 to retrieve all books and their authors. You desire to return additional authors as “et-­­al” after the first two authors.

Example 2.

```xml
for $b in document("books.xml")/bib/book
return
  if (count($b/author) <= 2) then $b
  else <book> { $b/@*, $b/title, $b/author[position() <= 2], <et-al/>, ...
      $b/publisher, $b/price } </book>
```

This query reads book data from a books.xml. If the author count is less than 2 or equal to 2 for each book, then the query returns the book straightly. Otherwise the query makes a new book element including all the original data, excepting that the query contains only the first two authors and attaches an et-al element. Position() function is returned only the first two authors. $b/@*, XPath expression, refers to all the attributes on $b.

3.3 XQuery functions and operators

XQuery contains a huge set of functions and operators. Table 2 shows frequently used built-in functions. You are able to describe your own and many engines provide custom extensions as well.

Table 2. Commonly used built-in functions
### Function

<table>
<thead>
<tr>
<th>Math:</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -, *, div, idiv, mod, =, !=, &lt;, &gt;, &lt;=, &gt;=, floor(), ceiling(), round(), count(), min(), max(), avg(), sum()</td>
</tr>
</tbody>
</table>

**Commentary**

Division is done using div rather than a slash because a slash indicates an XPath step expression. idiv is a special operator for integer-only division that returns an integer and ignores any remainder.

**Strings and Regular Expressions:**

| compare(), concat(), starts-with(), ends-with(), contains(), substring(), string-length(), substring-before(), substring-after(), normalize-space(), upper-case(), lower-case(), translate(), matches(), replace(), tokenize() |

**Commentary**

compare() dictates string ordering. translate() performs a special mapping of characters. matches(), replace(), and tokenize() use regular expressions to find, manipulate, and split string values.

### Date and Time:

| current-date(), current-time(), current-dateTime(), +, -, div eq, ne, lt, gt, le, gt |

**Commentary**

XQuery has many special types for date and time values such as duration, date-time, date, and time. On most you can do arithmetic and comparison operators as if they were numeric. The two-letter abbreviations stand for equal, not equal, less than, greater than, less than or equal, and greater than or equal.

### XML node and QNames:

| node-kind(), node-name(), base-uri() eq, ne, is, isnot, get-local-name-from-QName(), get-namespace-from-QName() |

**Commentary**

node-kind() returns the type of a node (i.e. "element"). node-name() returns the QName of the node, if it exists. base-uri() returns the URI this node is from.

Nodes and QName values can also be compared using eq and ne (for value comparison), or is and isnot (for identity comparison). deep-equal() compares two nodes based on their full recursive content.

The << operator returns true if the left operand precedes the right operand in document order. The >> operator is a following comparison.

### Sequences:

| item-at(), index-of(), empty(), exists(), distinct-nodes(), distinct-values(), insert(), remove(), subsequence(), unordered(), position(), last() |

**Commentary**

item-at() returns an item at a given position while index-of() attempts to find a position for a given item. empty() returns true if the sequence is empty and exists() returns true if it's not. distinct-nodes() returns a sequence with exactly identical nodes removed and distinct-values() returns a sequence with any duplicate atomic values removed. unordered() allows the query engine to optimize without preserving order. position() returns the position of the context item currently being processed. last() returns the index of the last item.

### Type Conversion:

| string(), data(), decimal(), boolean() |

**Commentary**

These functions return the node as the given type, where possible. data() returns the "typed value" of the node.

### Booleans:

| true(), false(), not() |

**Commentary**

There's no "true" or "false" keywords in XQuery but rather true() and false() functions. not() returns the boolean negation of its argument.

### Input:

| document(), input(), collection() |

**Commentary**

document() returns a document of nodes based on a URI parameter. collection() returns a collection based on a string parameter (perhaps multiple documents). input() returns a general engine-provided set of input nodes.

---

### 4. References

The contents of this chapter were quoted from the following lists.

- *X Is for XQuery*, Jason Hunter: http://www.oracle.com/technology/oramag/oracle/03-may/o33devxml.html
5. Useful Links and Books

- W3C XML Query (XQuery): http://www.w3.org/XML/Query
- XQuery Latest version: http://www.w3.org/TR/xquery/
- XQuery 1.0 and XPath 2.0 Functions and Operators: http://www.w3.org/TR/xpath-functions/
- XQuery 1.0 and XPath 2.0 Data Model (XDM): http://www.w3.org/TR/xpath-datamodel/
- XSLT 2.0 and XQuery 1.0 Serialization: http://www.w3.org/TR/xslt-xquery-serialization/
- XML Query Use Cases: http://www.w3.org/TR/xquery-use-cases/
- XML Query (XQuery) Requirements: http://www.w3.org/TR/xquery-requirements/
- XQuery: The XML Query Language, Michael Brundage, Addison-Wesley 2004

See Also

- XQuery Tutorial and Cookbook Wikibook This Wikibook has many small XQuery examples with links to working XQuery applications.

Exchanger XML Lite

Cladonia offers an xml editor at http://www.exchangerxml.com/ for free noncommercial use, and can be downloaded without registration.

This is a Java-based product that runs on all platforms including Windows, Linux, Mac OSX and UNIX.

(NOTE: If you need an XML editor for commercial use, you can get a free 30-day trial of Exchanger XML Professional at http://www.exchangerxml.com)

Single Entity in Exchanger XML Lite

The following directions will lead you step-by-step through doing the same project that is found in the XML - Managing Data Exchange/A single entity chapter.

Part One: Creating the Project Folder

1) Open Exchanger XML Lite
2) Click on:
   -Project
   -New Project : a "New Project" folder will appear in the project folder window
3) Type "TourGuide" over the "New Project" title to change the name of the new project to TourGuide.
Part Two: Creating the Schema File

1) Click on:
   - File
   - New
   - For Type
   - Scroll to "XML Schema Definition" and highlight it
   - OK

2) Exchanger automatically puts the beginning and ending tags in the file for you, however, for our example, delete those automatic tags, and copy and paste the following code into the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <!-- Tour Guide -->
  <!-- The tourGuide element is defined as a complex element type, i.e.
  it has embedded elements. -->
  <xsd:element name="tourGuide">
    <xsd:complexType>
      <xsd:sequence>
        <!-- The minimum number of times an element can occur is
         set using minOccurs (default is 1) and an unlimited
         number of times an element can occur maxOccurs="unbounded". -->
        <xsd:element name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- City -->
  <!-- City is declared an element of named complex type -->
  <!-- <xsd:complexType begins the declaration of a complex
  type, "cityDetails" identifies the complex type.
  This is NOT the name of the element. This complex type
  definition may be used in the declarations of more than one element. -->
  <xsd:complexType name="cityDetails">
    <!-- Schema element sequence {13} specifies that the child elements
    must appear in the order specified. -->
    <xsd:sequence>
      <!-- <xsd:element begins the declaration of an element of simple
      type, "cityName" is the name of the element being declared (in
      the XML document it will look something like: <cityName> )
      and "xsd:string" is the data type of the element. -->
      <xsd:element name="cityName" type="xsd:string"/>
      <xsd:element name="adminUnit" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="population" type="xsd:integer"/>
      <xsd:element name="area" type="xsd:integer"/>
      <xsd:element name="elevation" type="xsd:integer"/>
      <xsd:element name="longitude" type="xsd:decimal"/>
      <xsd:element name="latitude" type="xsd:decimal"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="history" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

3) Click on the GREEN CHECK to Validate, and the BROWN CHECK to check for Well-Formedness. These can be found on the toolbar:

4) Click on:
Part Three: Creating the Style Sheet

1) Click on:

- File
- New
- For Type
- Scroll to "XML StyleSheet Language" and highlight it
- OK

2) Delete any automatic tags that appear, and cut and paste the following code into the file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!doctype xsl:stylesheet public "-//W3C//DTD XSLT 1.0//EN" "http://www.w3.org/TR/xsl/1.0.dtd">
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Tour Guide</title>
      </head>
      <body>
        <h2>Cities</h2>
        <xsl:apply-templates select="tourGuide"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="tourGuide">
    <xsl:for-each select="city">
      <xsl:text>City: </xsl:text>
      <xsl:value-of select="cityName"/>
      <br/>
      <xsl:text>Population: </xsl:text>
      <xsl:value-of select="population"/>
      <br/>
      <xsl:text>Country: </xsl:text>
      <xsl:value-of select="country"/>
      <br/>
      <br/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

3) Click on the GREEN CHECK to Validate, and the BROWN CHECK to check for Well-Formedness. (NOTE: Be sure to eliminate any "white space" before the text that you paste, or you may have an error when validating.)

4) Click on:

- File
- Save As
- "city.xsl"

5) Right Click on:

- "TourGuide" project folder
- Add File
- "city.xsl"
- open

(Note: Now the project "TourGuide" contains two files, "city.xsd", and "city.xsl").

Part Four: Creating the XML File

1) Click on:

- File
- Save
2) Delete any automatic tags that appear and copy and paste the following code:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="city.xsl" type="text/xsl"/>
<!–The following declaration identifies the root element of the document
TourGuide and the schema file (city.xsd) using xsi:schemaLocation--> 
<tourGuide>
    <xsi:nsSchemaLocation http://www.w3.org/2001/XMLSchema-instance" xs://"city.xsd"
    xsi:namespacePrefix="city"> 
    <!--The definition of the first city--> 
    <city>
        <cityName>Belmopan</cityName>
        <adminUnit>Cayo</adminUnit>
        <country>Belize</country>
        <population>11100</population>
        <area>5</area>
        <elevation>130</elevation>
        <longitude>88.44</longitude>
        <latitude>17.27</latitude>
        <description>Belmopan is the capital of Belize</description>
        <history>Belmopan was established following the devastation of the former capital, Belize City, by Hurricane Hattie in 1965. High ground and open space influenced the choice and groundbreaking began in 1966. By 1978 most government offices and operations had already moved to the new location.
    </history>
    </city>
    <!--The definition of the second city--> 
    <city>
        <cityName>Kuala Lumpur</cityName>
        <adminUnit>Malaysia</adminUnit>
        <country>Malaysia</country>
        <population>1448600</population>
        <area>243</area>
        <elevation>111</elevation>
        <longitude>101.71</longitude>
        <latitude>3.16</latitude>
        <description>Kuala Lumpur is the capital of Malaysia and the largest city in the nation</description>
        <history>The city was founded in 1857 by Chinese tin miners and preceded Klang. In 1888 the British government transferred their headquarters from Klang to Kuala Lumpur, and in 1896 it became the capital of Malaysia.
    </history>
    </city>
</tourGuide>
```

3) Click on the GREEN CHECK to Validate, and the BROWN CHECK to check for Well-Formedness. (NOTE: Be sure to eliminate any "white space" before the text that you paste, or you may have an error when validating.)

4) Click on:

- File
  - Save As
  - city.xml

5) Right click on:

- TourGuide
  - Add File
  - "city.xml"

(Note: New project "TourGuide" should contain three files, "city.xsd", "city.xsl", and "city.xml".)

Part Five: Executing your code

1) Open the city.xml file.

2) Click on:

- Transform
  - Execute Simple XSLT
  - Current Document
  - OK

- XSL Input
  - From URL
  - pick city.xsl
  - open

- Use Default Processor
  - OK

(Note: the window should say "Transformation Complete"

Now you may close this window and follow step 3 to get the results.
3) Click on:

- Tools
- Start Browser

Note: Results should look like this:

Overview

ODBC is the acronym for the oft used API Open Database Connectivity. Many applications and application programmers use ODBC in order to access relational databases, such as SQL and Microsoft Access, and to manipulate the data within the databases. Specifically, JDBC (Java Database Connectivity), which is based on ODBC, is the API used by applications developed in Java to perform these various tasks. Moreover, JDBC is now capable of handling advanced datatypes in SQL which in turn becomes useful when dealing with XML. Also, JDBC has within it the ability to actually create XML data. Furthermore, the use of JAXP (Java API for XML Processing) along with JDBC provides yet another way of manipulating and using relational databases and XML. In any event, there are multiple ways to use the JDBC API with XML.

JDBC and XML Documents

Many Java Applications written today will more than likely interact with an SQL database (or a relational database, but for the sake of uniformity, we will work with SQL.) Depending on the intent of the application, there may be the case of actually storing an XML document for display or for manipulation. Whatever the case, JDBC now supports all datatypes defined in the SQL:1999 specification. One of these datatypes is the CLOB (character large object) datatype. This datatype is perfect for storing XML documents. This is one way XML and the JDBC API works with each other.

JDBC and XML Production

One of the more interesting things about JDBC is that it can be used to gather MetaData. Meta-data is nothing more than data about data. From an XML standpoint, this is very useful because we can create XML data on the fly with nothing more than a table name. The class that makes this possible is java.sql.ResultSetMetaData. Consequently this class is a part of the JDBC API.

JDBC and JAXP

Another intriguing way of dealing with XML objects is within the JAXP (Java API for XML Processing). JAXP and JDBC together provide an infrastructure for developing applications using XML and SQL.

Whenever XML instances in applications are dealt with, an XML parser is a good tool to use. The XML parser turns the XML document into an object or something the application can use. Specifically, Document Object Model (DOM) takes and XML instance and converts it into a tree. This specific parser can be found in the JAXP API. You may then store the parsed object in an SQL database for future use. This may open up many ideas of how one may use JAXP and JDBC together when an issue presents itself of dealing with XML and SQL.

References

- [http://www.xml.com](http://www.xml.com)
- [http://java.sun.com/xml](http://java.sun.com/xml)
What Is XForms?

Forms are an important part of many web applications today. An HTML form makes it possible for web applications to accept input from a user. Web users now do complex transactions that are starting to exceed the limitations of standard HTML forms. XForms is the next generation of HTML forms and is richer and more flexible than HTML forms.

XForms uses XML for data definition and HTML or XHTML for data display. XForms separates the data logic of a form from its presentation. Separating data from presentation makes XForms device independent, because the data model can be used for all devices. The presentation can be customized for different user interfaces, like mobile phones and handheld devices and can provide interactivity between such devices. It is also possible to add XForms elements directly into other XML applications like VoiceXML (speaking web data), WML (Wireless Markup Language), and SVG (Scalable Vector Graphics).

The Purpose of XForms

XForms is the separation of purpose from presentation. For example, the purpose of a questionnaire application is to collect information about the user. This is done by creating a presentation that allows the user to provide the required information. Web applications typically render such a presentation as an interactive document that is continuously updated during user interaction. By separating the purpose from its presentation, XForms enables the binding of different interactions to a single model.

The Main Aspects of XForms

The XForms model defines what the form is, what data it contains, and what it should do. The XForms user interface defines the input fields and how they should be displayed. The XForms Submit Protocol defines how XForms send and receive data, including the ability to suspend and resume the completion of a form.

XForms is "instance data", an internal representation of the data mapped to the familiar "form controls". Instance data is based on XML and defined in terms of XPath’s internal tree representation and processing of XML.

The XForms Framework

With XForms, input data is described in two different parts:

1. XForm model
2. XForm user interface

The XForms Model

The XForm model defines what the form is, what data it contains, and what it should do.

The data model is an instance (a template) of an XML document. The XForms model defines a data model inside a <model> element:

```xml
<model>
  <instance>
    <person>
      <fname/>
      <lname/>
    </person>
  </instance>
  <submission id="form1" action="submit.asp" method="get"/>
</model>
```

From the example above, you can see that the XForms model uses an <instance> element to define the XML template for data to be collected, and a <submission> element to describe how to submit the data.

The XForms model does not say anything about the visual part of the form (the user interface).

The <instance> Element

The data collected by XForms is expressed as XML instance data. XForms is always collecting data for an XML document. The <instance> element in the XForms model defines the XML document.

In the example above the "data instance" (the XML document) the form is collecting data for looks like this:

```xml
<person>
  <fname/>
</person>
```
After collecting the data, the XML document might look like this:

```xml
<person>
  <fname>Jim</fname>
  <lname>Jones</lname>
</person>
```

The `<submission>` Element

The XForms model uses a `<submission>` element to describe how to submit the data. The `<submission>` element defines a form and how it should be submitted. In the example above, the `id="form1"` attribute identifies the form, the `action="submit.asp"` attribute defines the URL to where the form should be submitted, and the `method="get"` attribute defines the method to use when submitting the data.

The following diagram shows how the XForm model has the capability to work with a variety of user interfaces.

The XForms User Interface

The XForms user interface is used to display and input the data. The user interface elements of XForms are called controls (or input controls):

```xml
<input ref="fname"><label>First Name</label></input>
<input ref="lname"><label>Last Name</label></input>
<submit submission="form1"><label>Submit</label></submit>
```

In the example above the two `<input>` elements define two input fields. The `ref="fname"` and `ref="lname"` attributes point to the `<fname>` and `<lname>` elements in the XForms model. The `<submit>` element has a `submission="form1"` attribute which refers to the `<submission>` element in the XForms model. A submit element is usually displayed as a button. Notice the `<label>` elements in the example. With XForms every input control element has a required `<label>` element.

Putting Everything Together

XForms has to run inside another XML document. It could run inside XHTML 1.0, and it will run inside XHTML 2.0. If we put it all together, the document will look like this:

```xml
<xforms>
  <model>
    <instance>
      <person>
        <fname/>
        <lname/>
      </person>
    </instance>
    <submission id="form1" action="submit.asp" method="get"/>
  </model>
</xforms>
```

The XForms Processor

An XForms Processor built into the browser will be responsible for submitting the XForms data to a target. The data can be submitted as XML and could look something like this:

```xml
<person>
  <fname>Jim</fname>
  <lname>Jones</lname>
</person>
```

Or it can be submitted as text, looking something like this:

```text
fname=Jim;lname=Jones
```

The XForms Namespace

The official namespace for XForms is: http://www.w3.org/2002/xforms. If you want to use XForms in HTML (or XHTML 1.0), you should declare all XForms elements with an XForms namespace. XForms is expected to be a standard part of XHTML 2.0, eliminating the need for the XForms namespace.

An XForms Example

Take a look at this document using XForms:
The Form Controls

The components of the form that deal with data entry and display are referred to as the form controls or user interface controls. XForms defines a comprehensive set of device-neutral, platform-independent form controls. For each element of data defined in the model, a form control defines its appearance via the client. These controls can be combined with stylesheets to provide sophisticated form displays.

<table>
<thead>
<tr>
<th>XForms form control</th>
<th>Closest XHTML equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;input&gt;</td>
<td>&lt;input type=&quot;text&quot;&gt;</td>
<td>For entry of small amounts of text</td>
</tr>
<tr>
<td>&lt;textarea&gt;</td>
<td>&lt;textarea&gt;</td>
<td>For entry of large amounts of text</td>
</tr>
<tr>
<td>&lt;secret&gt;</td>
<td>&lt;textarea&gt;</td>
<td>For entry of large amounts of text</td>
</tr>
<tr>
<td>&lt;secret&gt;</td>
<td>&lt;input type=&quot;password&quot;&gt;</td>
<td>For entry of sensitive information</td>
</tr>
<tr>
<td>&lt;output&gt;</td>
<td>N/A</td>
<td>For inline display of any instance data</td>
</tr>
<tr>
<td>&lt;range&gt;</td>
<td>N/A</td>
<td>For smooth &quot;volume control&quot; selection of a value</td>
</tr>
<tr>
<td>&lt;upload&gt;</td>
<td>&lt;input type=&quot;file&quot;&gt;</td>
<td>For upload of file or device data</td>
</tr>
<tr>
<td>&lt;trigger&gt;</td>
<td>&lt;button&gt;</td>
<td>For activation of form events</td>
</tr>
<tr>
<td>&lt;submit&gt;</td>
<td>&lt;input type=&quot;submit&quot;&gt;</td>
<td>For submission of form data</td>
</tr>
<tr>
<td>&lt;select&gt;</td>
<td>&lt;select multiple=&quot;multiple&quot;&gt; or multiple &lt;input type=&quot;checkbox&quot;&gt;</td>
<td>For selection of zero, one, or many options</td>
</tr>
<tr>
<td>&lt;select1&gt;</td>
<td>&lt;select&gt; or multiple &lt;input type=&quot;radio&quot;&gt;</td>
<td>For selection of just one option among several</td>
</tr>
</tbody>
</table>

XForms Action

In the course of form processing, often some particular action needs to happen.

<table>
<thead>
<tr>
<th>XForms Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setfocus</td>
<td>Gives focus to a particular form control.</td>
</tr>
<tr>
<td>setvalue</td>
<td>Sets the value of a particular node.</td>
</tr>
<tr>
<td>message</td>
<td>Displays a message to the user.</td>
</tr>
<tr>
<td>send</td>
<td>Submits all or part of the instance data.</td>
</tr>
<tr>
<td>reset</td>
<td>Resets all or part of the instance data.</td>
</tr>
<tr>
<td>load</td>
<td>Opens a document in the same or a new window.</td>
</tr>
<tr>
<td>refresh</td>
<td>Refreshes the view of the instance data.</td>
</tr>
<tr>
<td>recalculate</td>
<td>Recalculates the instance data.</td>
</tr>
<tr>
<td>revalidate</td>
<td>Revalidates the instance data.</td>
</tr>
<tr>
<td>setindex</td>
<td>Navigates through a repeating sequence.</td>
</tr>
<tr>
<td>insert</td>
<td>Inserts a node from a repeating sequence.</td>
</tr>
<tr>
<td>delete</td>
<td>Removes a node from a repeating sequence.</td>
</tr>
<tr>
<td>toggle</td>
<td>Selects a case of a switch</td>
</tr>
<tr>
<td>dispatch</td>
<td>Dispatch an event.</td>
</tr>
</tbody>
</table>

XForms Methods

The XForms specification uses and builds upon XPath, which includes adding some method calls useful for forms: These can be called at any point where XPath is allowed. Additionally, implementations can support "extension functions" to provide additional functionality.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg()</td>
<td>Returns the arithmetic mean of the indicated nodes</td>
</tr>
<tr>
<td>min() and max()</td>
<td>Returns the minimum or maximum value of the indicated nodes</td>
</tr>
<tr>
<td>count-non-empty()</td>
<td>Returns the number of non-empty nodes</td>
</tr>
<tr>
<td>if()</td>
<td>Returns one of two strings depending on a Boolean value</td>
</tr>
<tr>
<td>index()</td>
<td>Indicates the current position in a repeating sequence</td>
</tr>
<tr>
<td>days-from-date()</td>
<td>Converts an XML Schema datatype into a number of days</td>
</tr>
<tr>
<td>seconds-from-datetime()</td>
<td>Converts an XML Schema datatype into a number of seconds</td>
</tr>
<tr>
<td>seconds()</td>
<td>Converts an XML Schema duration into a number of seconds</td>
</tr>
<tr>
<td>months()</td>
<td>Converts an XML Schema duration into a number of months</td>
</tr>
</tbody>
</table>
XML Web Voice

Learning Objectives

Upon completion of this chapter, you will

- understand the potential of XML based web delivery of audio text files
- understand the use of SSML (Synthetic Speech Markup Language), the XML subset of tags

The Problem:

How can text files of any content a user chooses be transferred to the user in such a form that the end user can listen to them on a piece of hardware that has mobile internet capability? An example would be someone listening to a news report of any topic or from any region just by converting text files which already exist on the internet to a synthesized voice for playing on a mobile device in an automobile.

Existing Technology: There are several technologies available that work with voice data. Voice XML provides a framework for transferring voice data between entities. It is used for interactive voice triggered tools. This technology is used extensively for phone menus and automated help by companies with customer service or other areas of high call volume needs.

Internet radio exists and provides a user with music or other programming that is broadcast throughout the internet. This programming is not up to the choice of the end user other than to select the internet station to listen to.

Software exists that can convert any text file into an audio file. Text files can be converted to audio files using software provided in the Windows or Mac operating systems or very inexpensive stand-alone software (an example is TextAloud). TextAloud allows a user to modify the voice, the pace of reading and other features. Free versions of it are available online. These systems can modify the voice in multiple ways to the personal taste of a user. These systems do not make the files available over the internet for users to search and listen to.

The Potential:

With the right combination of XML technology, mobile communication services and software/hardware that already exists the idea of internet radio could be opened to a much larger volume of content than currently exists. Most internet radio is in the form of music files and programmed radio content. The choices of internet radio could be extended to include any existing text file which would include news reports, government documents, educational materials and many forms of official records. A business example would be a travelling salesman briefing himself on a client’s buying history by listening to a file in his car on the way to a sales call with the customer. Another example includes language conversion software that already exists that could enable a person in a distant county to listen and learn about technology that is being developed somewhere else.

Requirements:

The technology would require three areas to come together to make the process work. 1. The XML technology must include a set of agreed upon XML tags for transferring files between content generators/distributors and users. 2. The mobile communications services must be able to deliver the data in a usable format to an end user system. 3. Hardware and software must be able to make use of the documents sent and play them for a user. Included in this is the further development of a voice processing browser.

The second and third requirements are outside the scope of this chapter on XML. However, work is being done on them. The W3C (World Wide Web Consortium) is currently working on the Mobile Web Initiative which would set some standards for software vendors, content providers, hardware (handset) manufacturers, browser developers and mobile service operators. One suggestion being considered is a maximum page weight of 10K (a typical magazine article fits within that range). The availability of advertising being embedded and what form it would take is under debate. Delivery protocol is expected to be http. The connection for a mobile device can be slow but the audio files do not have to stream. Current vendors involved include Nokia, Ericsson, HP, France Telecom and Opera.
The first requirement would include a set of XML tags that all text file content generators (such as news services, governments, educational institutions and official records generators) could use to generate files of their content. Thus their content could be accessed and stored in a searchable database and requested for downloading and playback at any time from anywhere that supports a mobile browser device.

The Existing Tag Set:

There is an existing set of XML tags called SSML (Synthesized Speech Markup Language). This set enables control of enough aspects of speech generation that a personable voice can be generated and manipulated by a user. A Text-to-Speech system uses the tags to take a text file and generate audible text in a voice.

Document Structure, Text Processing and Pronunciation Elements and Attributes:

**speak** - Root Element

- **xml:lang** - Attribute
  - Language (indicates the natural language of the file, such as "en-US"); this is preferred to be indicated only on the voice element so as to eliminate changes in a voice in the midst of a voice file.

- **xml:base** - Attribute
  - base URI Attribute (optional)

**EXAMPLE:**

```
<speak version="1.0"
   xmlns="http://www.w3.org/2001/10/synthesis"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
   http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
   xml:lang="en-US">
   ... the body ...
</speak>
```

**lexicon** - Element

- for pronunciation, (an empty element)

**meta** - Element

- (an empty element); includes a string that contains some information about the ensuing data; it can declare a content type of "http" in the case of a file that doesn't have generated header fields from the originating server.

**metadata** - Element

- can provide broader information about data as it accesses a metadata schema.

**p** - Element

- text structure, represents a paragraph. It can only contain the following elements: audio, break, emphasis, mark, phoneme, prosody, say-as, sub, s, voice.

**s** - Element

- text structure, element; represents a sentence. It can only contain the following elements: audio, break, emphasis, mark, phoneme, prosody, say-as, sub, s, voice.

**say-as** - Element
the only required one. The tag set may only contain text to be rendered by a voice synthesizer. This tag helps a browser to know more about the manner in which the enclosed text is to be voiced.

**format** - Attribute

- this attribute gives additional hints as to the rendering of voiced text.
- detail - Attribute
  - this attribute is for indicating the level of detail to be applied to voiced text. An example would be a special form of emphasis such as the reading of computer code in a block of text.

**Phoneme** - Element

- a pronunciation indicator for the text to speech engine. The engine does not render the contents of the tag, thus the tag can be empty. The attributes for the tag provide what the engine will use to help with language specific pronunciation factors.
- alphabet - attribute
  - for Phoneme, used to specify a particular version of an alphabet, optional
- ph - Attribute
  - a required attribute for phoneme, used to specify the string to be pronounced.

**EXAMPLE:**

```xml
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/10/synthesis http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
  xml:lang="en-US">
  <phoneme alphabet="ipa" ph="təmeiɾou̥">pomegranate</phoneme>
</speak>
```

**sub** - Element

- an element used to specify within its "alias" attribute the pronounced version of some written text that is between the tag set. Example:

**AARP**

Prosody and Style - prosody covers such things as tone, intonation, conversational pacing, pitch of voice, loudness, duration of sound, chunking (units of words, not necessarily sentences).

**voice** - Element

- indicates the type of voice to use, all the attributes are optional, however, not indicating any attributes at all is considered an error. The "lang" attribute takes precedence; all other attributes are equal.
- lang - attribute
  - for voice element, indicates the language for the voice.
- gender - Attribute
- age - Attribute
- variant - Attribute
- name - Attribute

**EXAMPLE:**

```xml
<voice gender="male">Show me a person without a goal</voice>
<voice gender="male" variant="1">and I'll show you a stock clerk.</voice>
<voice name="James">Show me a stock clerk with a goal and I'll show you someone who will change the world.</voice>
```

**emphasis** - Element

- contains text to be emphasized by the speech processor (with stress or intensity). It has one attribute:
- level - Attribute
  - indicating the degree of emphasis.

**EXAMPLE:**

Geniuses themselves don't talk about the gift of genius, they just talk about

```xml
<emphasis level="strong">hard work and long hours</emphasis>
```
The "emphasis" element can contain text and the following elements: audio - Element desc - Element

if the content is not speech then the "desc" tag should be used to describe the content. This description can be used in a text output for the hearing impaired.

break - Element emphasis - Element mark - Element phoneme - Element prosody - Element say-as - Element sub - Element voice - Element

break - Element

wherever the element is used between words it indicates a pause in the reading of the text; attributes are: "strength" with values of: none (meaning no pause even if the system would normally put one there), x-weak, weak, medium, strong, x-strong; "time" with values of either milliseconds: 250ms or seconds: 2s.

prosody - Element

controls the pitch, speaking rate and volume of a generated voice. Attributes are optional but it is considered an error if no attributes are set.

pitch - Attribute
contour - Attribute
range - Attribute
rate - Attribute
duration - Attribute
volume - Attribute

Other elements that allow the insertion of audio files in addition to generated voice content.

audio - Element

may be empty but if it contains anything it should be the text that the speech generator could convert to a voice in place of the audio file.

EXAMPLE:

<audio src="JCPennyQuote.au">Every business is built on friendship.</audio>

mark - Element

an empty tag that places a named marker into the content. When the processor reaches a "mark" element one of two things happens. One, the processor is provided with the info to retrieve the desired position in the content. Two, an event is issued that includes the content at the desired position. It has one attribute which is:

name - Attribute

desc - Element

Potential Future of XML Web Audio:

Additional tags could be introduced to contain dates, titles of files, authors, originating language and other metadata about the files. Expanding the set of existing tags would enable the files to be stored and searched in databases using multiple methods. They would enable storing of data related to the actual text/audio files that would be valuable to potential users. A user could search based on originating date of the file, the originating country of the file and subject or title of files.

Conclusion:

Using SSML, a subset of XML, audio files can be generated from any text file such as news reports, government documents, educational materials or official records. This content could be delivered via mobile communication services and over the web. The files could be played on mobile browser devices. This could constitute a much larger market for internet radio than the strictly music or programmed content form it exists in today. This could generate many uses for on-demand access to many sources of information for travelling users.
Introduction

OpenOffice.org is exactly what its title suggests: an open source office applications suite. It is based on the source of Sun Microsystems’ StarOffice, which was donated to the open source community in 2000. OpenOffice.org will read and save files in several formats used by other office applications, but its default format is OpenDocument, which is an XML format standardized by OASIS (Organization for the Advancement of Structured Information Standards). Because of these two factors, an open source editor and XML-based files, OpenOffice.org is poised to be of great importance in the very near future as the trend continues for national governments, particularly in the European Union, to require that all electronic government documents be saved in an open source format.

OpenDocument Format

As stated in the introduction, OpenDocument is an XML format standardized by OASIS. An OpenDocument file takes the form of a compressed zip archive with one of the following extensions:

- .odt (text)
- .ott (text template)
- .odm (master document)
- .oth (HTML template)
- .ods (spreadsheet)
- .ots (spreadsheet template)
- .odg (drawing)
- .otg (drawing template)
- .odp (presentation)
- .otp (presentation template)
- .odf (formula)
- .odb (database)

(Note that all of these extensions are version 2.0 only)

The zip archive contains the following files and directories (from OpenOffice.org Help documentation):

- The actual text of the document is stored in `content.xml`. By default this is a stripped-down version of the document that leaves out formatting elements such as indentation or line breaks in order to streamline saving and opening the document.

**content.xml Example**

```xml
<office:document-content namespace-declarations
office:version="1.0"
office:class="document-type">
  <office:scripts/>
  <office:font-decls>
    <!−− font specifications −−>
  </office:font-decls>
  <office:styles>
    <office:automatic-styles>
      <!−− style information −−>
    </office:automatic-styles>
  </office:styles>
  <office:body>
    <office:document-type>
      <!−− actual content here −−>
    </office:document-type>
  </office:body>
</office:document-content>
```

- `meta.xml` contains the meta information of the document, which can be edited in File – Properties. If a document is saved with a password, meta.xml will not be encrypted.

**meta.xml Example**

```xml
<?xml version="1.0" encoding="UTF-8"?>
office:meta>
  <meta:generator>OpenOffice.org/2.08Win32 OpenOffice.org_project/680M180Build-6999</meta:generator>
  <meta:initial-creator/><meta:creation-date>2006-03-27T19:17:57</meta:creation-date>
  <meta:creator Name="DC:creator">Creator Name</meta:creator>
  <meta:creation-date>2006-03-27T20:58:06</meta:creation-date>
  <dc:creator>Create Name</dc:creator>
  <dc:date>2006-03-27T20:58:06</dc:date>
  <dc:language>en-US</dc:language>
  <meta:editing-cycles><meta:edit-count>0</meta:edit-count>
  <meta:editing-duration>PT0M0S</meta:editing-duration>
  <meta:user-defined meta:name="Info 1"/>
  <meta:user-defined meta:name="Info 2"/>
  <meta:user-defined meta:name="Info 3"/>
  <meta:user-defined meta:name="Info 4"/>
</office:meta>
```
**settings.xml** contains further information about the settings for this document.

**settings.xml Example**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  <office:settings office:system="default-module" config:name="oo:default-settings">
    <config:config-item-set config:name="oo:view-settings">
      <config:config-item config:name="ViewAreaTop" config:type="int">635</config-item>
      <config:config-item config:name="ViewAreaLeft" config:type="int">32582</config-item>
      <config:config-item config:name="ViewAreaRight" config:type="int">"</config-item>
      <config:config-item config:name="ViewAreaBottom" config:type="int">43486</config-item>
    </config:config-item-set>
    <config:config-item config:name="ShowRedlineChanges" config:type="boolean">true</config-item>
    <config:config-item config:name="RedlineProtectionKey" config:type="base64Binary" />
    <config:config-item config:name="PrintPaperFromSetup" config:type="boolean">false</config-item>
    <config:config-item config:name="UpdateFromTemplate" config:type="boolean">false</config-item>
    <config:config-item config:name="PrintSingleJobs" config:type="boolean">false</config-item>
    <config:config-item config:name="IgnoreFirstLineIndentInNumbering" config:type="boolean">false</config-item>
    <config:config-item config:name="LinkUpdateMode" config:type="short">1</config-item>
    <config:config-item config:name="ZoomFactor" config:type="short">100</config-item>
    <config:config-item config:name="VisibleRight" config:type="int">25850</config-item>
    <config:config-item config:name="VisibleTop" config:type="int">635</config-item>
    <config:config-item config:name="VisibleLeft" config:type="int">0</config-item>
    <config:config-item config:name="ViewLeft" config:type="int">17549</config-item>
    <config:config-item config:name="ViewId" config:type="string">view2</config-item>
    <config:config-item config:name="ShowRedlineChanges" config:type="boolean">true</config-item>
    <config:config-item config:name="ViewAreaHeight" config:type="int">14818</config-item>
    <config:config-item config:name="ViewAreaTop" config:type="int">635</config-item>
    <config:config-item config:name="AddRedlineFormatting" config:type="boolean">false</config-item>
    <config:config-item config:name="AddRedlineLinking" config:type="boolean">false</config-item>
    <config:config-item config:name="AddRedlineTracking" config:type="boolean">false</config-item>
    <config:config-item config:name="AddRedlineTracking" config:type="boolean">false</config-item>
    <config:config-item config:name="AddRedlineTracking" config:type="boolean">false</config-item>
    <config:config-item config:name="AddRedlineTracking" config:type="boolean">false</config-item>
  </office:settings>
</office:office>
```

**styles.xml** contains the styles applied to the document that can be seen in the Styles and Formatting window.

**styles.xml Example**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  <office:styles>
    <office:style-name style:name="Tahoe1" style:family="tahoe"/>
    <office:style-name style:name="Arial Unicode MS" style:family="Arial Unicode MS" style:font-pitch="variable"/>
    <office:style-name style:name="Arial" style:font-family-generic="swiss" style:font-pitch="variable"/>
    <office:style-name style:name="Tahoe2" style:family="tahoe"/>
  </office:styles>
</office:office>
```
manifest.xml in the meta-inf directory describes the structure of the XML file.

manifest.xml Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0"
  xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:fo:1.0"
  xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
  xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"
  xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"
  xmlns:graphic="urn:oasis:names:tc:opendocument:xmlns:graphic:1.0"
  xmlns:presentation="urn:oasis:names:tc:opendocument:xmlns:presentation:1.0"
  xmlns:svg="urn:oasis:names:tc:opendocument:xmlns:svg:1.0"
  xmlns:svgx="urn:oasis:names:tc:opendocument:xmlns:svgx:1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  <manifest:manifest>
    <manifest:file-entry manifest:full-path="/"/>
    <manifest:file-entry manifest:full-path="/settings.xml"/>
    <manifest:file-entry manifest:full-path="/meta.xml"/>
    <manifest:file-entry manifest:full-path="/styles.xml"/>
    <manifest:file-entry manifest:full-path="/pictures/"
      manifest:media-type="image/jpeg"/>
    <manifest:file-entry manifest:full-path="/thumbnails/thumbnail.png"/>
    <manifest:file-entry manifest:full-path="/thumbnails/thumbnail.png"/>
    <manifest:file-entry manifest:full-path="/configurations2/"/>
    <manifest:file-entry manifest:full-path="/configurations2/"/>
    <manifest:file-entry manifest:full-path="/configuration.xml"/>
    <manifest:file-entry manifest:full-path="/configuration.xml"/>
  </manifest:manifest>
</manifest>
```

Open Office Capabilities

OpenOffice contains WRITER which is OpenOffice.org’s word processor. CALC is a powerful spreadsheet that has all the tools needed to calculate, analyze, summarize, and present data. IMPRESS is a fast and powerful way to create effective multimedia presentations. DRAW will produce everything from simple diagrams to dynamic 3D illustrations and special effects. New to Version 2, BASE enables you to manipulate database data seamlessly within OpenOffice.org. It allows the creation and modification of tables, forms, queries, and reports.

Download at http://download.openoffice.org/

The Future of OpenDocument

Government and business leaders alike are beginning to realize the importance of open source document formats and to act upon this realization. In July 2005, Norway’s Minister of Modernization, Morton A. Meyer, presented a plan for information technology in Norway called “eNorge—the digital leap,” in which open standards and open source are addressed. Meyer’s objectives for open standards and open source are as follows:

- Within 2009 all new ICT- and information systems in the public sector shall use open standards.
- Within 2006 a set of management/administrative standards for data and document exchange should be established.
- Within 2006 all operations in the public sector should have introduced plans for how they will use open standards, service oriented architecture and open source.
- Within 2008 all data and document exchange in the public sector shall satisfy the management/administrative standards.
- Within 2008 all public forms shall be built on a common interface.

Also, Meyer said “Proprietary formats will no longer be acceptable in communication between citizens and government.”

In the US, IBM's Vice President of Standards and Open Source, Bob Sutor, made the following recommendations to users in his blog as part of an “Open Document Commitment to Action”:

- Insist today that the provider of your office applications (word processor, spreadsheet, presentation software) is committed to support the OASIS OpenDocument Format for Office Applications standard in their products by January 1, 2007.
- Insist today that the office applications you deploy allow users to easily set the OASIS OpenDocument standard as the default "save" format for your documents. That is, you should not have to go to a lot of trouble to avoid using proprietary formats.
- Get a commitment from your office applications provider to join and contribute to the OASIS OpenDocument standard technical committee.
- Ask your CIO when you will be able to use office applications that support the OASIS OpenDocument standard.
- Ask your local and federal governments when they will be supporting the OASIS OpenDocument standard.
- Insist that any XML document format you use is not encumbered by proprietary extensions, and that the format is freely available for anyone to implement without restrictions, including open source communities that use a GPL license. Ensure that if implementers must accept a license covering the format, the license is clear and unambiguous on these important issues.
- "They are your documents: you should be able to do whatever you want with them, whenever you want, with whatever application you wish to use"

The implications are clear. The computing world is moving toward open source software and file types for information exchange, especially in business and government, and OpenOffice.org, with its XML-based format, stands to be at the front of the pack.

Additional Links

http://www.sun.com/software/star/staroffice/index.jsp
http://www.koffice.org/

Works Cited


XMLRPC - Serializing and Deserializing Datatypes


It works well for its intended use. Well, unless you are coding in PHP and using the built-in, undocumented functions to generate XMLRPC. You may have to build your own, just to be on the safe side.

Just a quick example of what it looks like. In xmlrpc, a <struct> is what we would call in php an 'associative array', whereas an <array> is a 'list' or an array with only numerical indices (or in some languages, no indices). A complex array datatype with the following structure:

```xml
Array
  [0] => dogs
  [1] => cats
  [animals] => Array(
    [0] => FALSE
    [1] => little_dogs
    [2] => little_cats
    [3] => 5
    [4] => 2.3
    [5] => 1
  ),
```

Becomes the following XMLRPC:

```xml
<?xml version="1.0" encoding="utf-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>0</name>
            <value>
              <string>dogs</string>
            </value>
          </member>
          <member>
            <name>1</name>
            <value>
              <string>cats</string>
            </value>
          </member>
          <member>
            <name>animals</name>
            <value>
              <array>
                <data>
                  <value><boolean>0</boolean></value>
                  <value><string>little_dogs</string></value>
                  <value><string>little_cats</string></value>
                  <value><i4>5</i4></value>
                  <value><double>2.3</double></value>
                  <value><boolean>1</boolean></value>
                </data>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

Appendices

Editing Chapters

You simply click on the "edit this page" tab at the top of any page you want to edit. You can also just edit a section or subsection - any place you see a little 'edit' to the far right above a line, click on it to edit that section. The thoughts behind the editing process of the chapters will be documented in the "discussion" tab of each chapter - go there to record your intentions for your assigned chapter.

If you would like to make any suggestions to the editorial board, we welcome them. At the top of the XML: Managing Data Exchange contents page is a tab to "discussion" follow that link and edit the page with your comments, be sure to follow the directions for entries.

- Use wiki code, not HTML code when you edit a page.
- how to make a link: [[XML: Managing Data Exchange/NameOfThePage(on the page)|NameOfThePage(displayed as the link)]]
- no plagiarism: please use citations and references to books, articles, and websites that have helped you contribute to the book.
- All figures and graphics should be prefixed with 'xml' to avoid accidentally overwriting another file (e.g., xmdmcity defines the data model for city).
- Where possible use Portable Network Graphics (PNG) (http://www.libpng.org/pub/png/) for graphics because of its superiority to GIF and JPEG. Also,
Consistency guidelines

- developed to create a consistent look and feel to the book

Layout

- Chapter summary required

The bottom of each chapter should include a chapter summary highlighting the key points of the chapter. Follow the formatting guideline detailed later on this page.

- Bold new words

As new concepts and terminology are introduced in each chapter, make sure that they are defined first before using them; so that the reader understands what the words and ideas represent. Bold the first occurrence of the word (or preceding the definition).

- Use the sentence format for titles

Capitalize the first letter of a title and allow the rest of the letters to be lower case, as in this title.

- Avoid the use of underline and all capital words

- Tables, figures, and code references

For consistency, label all tables, figures (e.g. charts and photos), and code examples as "exhibits." Place the label for exhibits at the top of the exhibit, flush left, and enumerate the exhibits, followed by a brief caption. Examples: "Exhibit 1: XML data types" and "Exhibit 2: Schema code example"

- Concept capsules and breathing room

Think of your chapter as a data model or a large steak. A useful data model is broken down into its smallest attributes that require recording and maintaining for future analysis. A large steak is best eaten in bite size pieces to prevent choking and promote good digestion. As you revise your chapter consider the basic elements of your topic; break your topic down and present each element in a section - so that it is easy for the reader to follow and understand each part. Feel free to use bullets to represent lists.

To continue with the steak analogy, it is also good to eat slowly, taking breaths between each bite. Allow for spaces between each subsection and greater spaces between each section. This opens up the text so that it does not appear to dense and compact - and therefore intimidating.

- NetBeans (or other XML editor) Examples

Scroll down the contents page to "Appendices". You will see a link to the chapter "Using an XML editor". When you open this link - go to the edit this page tab and insert a link under the NetBeans heading with the name of your chapter. This is the format for the link: [XML: Managing Data Exchange/ChapterName(NetBeans[ChapterName])] After you save your edit, the XML Editor page will contain a link to your new NetBeans page. It will appear red because no information has been posted to that page yet. Click on the link and paste all the NetBeans information from your chapter into the edit box. In your assigned chapter, at the places where there were NetBeans information (that you just cut and paste into the XML editor page) - put a link to the XML editor - your Chapter page.

- Help:Wikibooks <= this link will provide several wiki guidelines - this page is also accessible from the 'help' link in the navigation box on the left bar of every one of these Wikibook pages. It contains information on inserting pictures, creating tables using wiki language (which is much easier to write and read).

This help page also has a link to a "sandbox." This is a blank page that allows you to experiment with wiki without affecting a real page.

Code examples

- Make all code examples a uniform color (black)—some are all black; some are multicolored. Please change the ones that are multicolored to all black.

- HTML code is lower case.

- Keep the comments in the code, but put a space above and below each comment.

- Include the file name of the code in a comments tag. If you are the author of the code, include your name and the date you authored the code in the comments. See example:

```
<!‐‐
Document   : city.xsd
Created on : March 1, 2005
Author     : Tim Jones
‐‐>
```

- Keep all the example codes in the chapter on the same topic—the TOURGUIDE. If there are examples in your chapter that deviate from this theme, please change them to show examples from TOURGUIDE.

- If your code contains HTML or the XML declaration then you will want to put the tags <pre> ... </pre>

- In order to get those 'pre' tags to show up I had to use the tags <nowiki> ... </nowiki>

If you run into trouble while inserting code, one of the two tags mentioned above might be able to get you out.

- You do not have to create a table around samples of a code example; usually just putting a few spaces in front of each line will create the dashed blue lined box

Spelling

- Use the spelling "stylesheet" instead of "style sheet."

- Use the spelling "opening tag" instead of "opening-tag."

- Check your wiki text with a spellchecker (e.g. copy and paste in a text editor that supports spell checking).
Create two links at the bottom of your chapter, one to an Exercises page and one to the Answers page already made (you can find this answer page link at the bottom of the contents page).

In the Exercises page, cut and paste your chapter exercises into a new page. An easy way to create a new page is simply to go to your main chapter page, type "exercises" at the end of the URL, and hit enter. If the page does not exist, Wikipedia will ask you to create the page by going to "edit this page."

At the top and bottom of the Exercises page, provide a link to the chapter and the Answers page—and vice versa with the Answers page.

**Elements of Style - Principles of Composition**

*taken from the Elements of Style* - by William Strunk and E.B. White

- **Make the paragraph the unit of composition** - The subject will need to be broken down into topics, and topics will need to be broken down into concepts and subconcepts. Each concept should be explored within its own paragraph.
- **Put statements in positive form** - "Make definitive assertions. Avoid tame, hesitating, noncommittal language. Use the word 'not' as a means of denial or in antithesis, never as means of evasion."
- **Use definite, specific, concrete language** - "Prefer the specific to the general, the definite to the vague, the concrete to the abstract."
- **Omit needless words** - "A sentence should contain no unnecessary words, a paragraph no unnecessary sentences. Many expressions violate this principle: 'the fact that,' he is a man who ... - be ..."
- **Avoid a succession of loose sentences**
- **Express coordinate ideas in similar form**
- **Keep related words together**
- **Place the emphatic words of a sentence at the end**

**Source code**

For inserting source code (e.g., XML or Java) into the book, use the following format:

```xml
<country>
  <code>au</code>
  <country>Australia</country>
  <flag>au.gif</flag>
</country>
```

Use the following wiki code:

```wiki
| style="background:linen; border: 1px solid black; padding: 1em; width: 100%;"
| A chapter summary
```

**Section summary**

Section summaries should appear in the following format:

```
| A chapter summary
```

Use the following wiki code:

```wiki
| style="background: lightyellow; border: 1px solid black; padding: 1em; width: 100%;"
| A story
```

**Story**

Stories/case studies should appear in the following format:

```
| A story
```

**References**
HTML tables can be converted to Wiki format using an online tool (http://www.uni-bonn.de/~manfear/html2wiki-tables.php/wiki).

NetBeans

Using an XML Editor: NetBeans: Single - Entity

XML document:

2. Save your file as ‘city’.
4. Next, specify your schema location (Go to Browse, then select the schema just created) and root element name (tourGuide).
5. Copy and paste the above code into the NetBeans editor and save.
6. Check the XML document to make sure it is well-formed (ALT-F9).

To validate an XML document:

1. Right-click the document's node and choose Validate XML.

If the XML file, city.xml, conforms to the XML schema provided, city.xsd, your Output Window should look like the following:

Open this file, city.xml, in NetBeans.

Summary: Being a file which contains XML code and syntax, the XML document is an entity of neatly organized elements and subelements.

NetBeans Tips

The prefix "xsi" for attribute "xsi:noNamespaceSchemaLocation" associated with an element type "Listing" is not bound.

-cvc-elt.1: Cannot find the declaration of element 'Listing'.

-cvc-complex-type.2.4.a: Invalid content was found starting with element 'MuseumName'. One of '{"":Museum}' is expected. [5]

The element type "MuseumName" must be terminated by the matching end-tag "</MuseumName>".

XML Schema

Any XML document, including XML schema and XML stylesheet documents, can be checked to make sure they are “well-formed”. An XML document is said to be well-formed if it adheres to the XML syntax rules, as outlined in the Chapter 1, such as there must be exactly one root element, every start tag must have an end tag. The well-formedness of an XML documents can be checked using an XML editor software, in this case, the NetBeans. If any errors are detected, they are listed along with the line number in the Output window. This location is not necessarily the line where the error occurred. By double-clicking on any error message, you can go to the line in the XML document where the error was detected.

Use NetBeans to create a new project:

1. Go to File -> New Project

Use NetBeans to create the above XML schema:

2. Save your file as ‘city’ (this gives the file an extension of .xsd).
3. Copy and Paste the above schema into city.xsd and save
To check an XML document to see if it is well-formed:

1. Right-click the document's node (under files, city.xsd) and choose Check XML schema or press Alt-F9.

If the XML schema, city.xsd, is well-formed, your Output Window should look like this:

```
Summary: The XML schema shows the structure and elements of an XML document.
```

Netbeans Tips

Element type "xsd:element" must be followed by either attribute specifications, ">" or "/>".

- Caused by lack of ">" or "/>" symbol at the end of either the specified line or the line there after.

The value of attribute "name" associated with an element type "xsd:complexType" must not contain the '<' character.

- Often caused by lack of closing quotation around the variable name on either the specified line or the line there after.

src-resolve.4.2: Error resolving component 'xsd:string'...

- Caused by invalid "type", most likely misspelled. Can be for any "type", not just "string."

The element type "xsd:sequence" must be terminated by the matching end-tag "</xsd:sequence>".

- Caused because </xsd:sequence> is not found in order to close the statement.
- Also caused because the closing statements are out of order as they at the beginning of the Schema.

XML Stylesheet

Use NetBeans to create the above XML stylesheet:

1. Go to File -> New File -> XML -> XSL Stylesheet
2. Save your file as 'city'
3. Copy and paste the above code into the NetBeans editor and save.
4. Check the stylesheet to make sure it is well-formed (ALT-F9).

Note: The city example files (city.xsd, city.xml and city.xsl) can be created using a text editor, and the results are displayed correctly in a compatible web browser such as Mozilla 1.6. You will need to store all of the files in the same directory and insert the following line after line 1 of city.xml:

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

This associates the XSL Stylesheet with the XML file containing the city data. Also, If you are using Netbeans, be sure to take off the xmlns line in the beginning which looks like xmlns=http://xml.netbeans.org/examples/targetNS, otherwise it will not display all data.

Netbeans Tips

The element type "xsl:text" must be terminated by the matching end-tag "</xsl:text>".

- Caused by either lack of end-tag or misspelling of the end-tag.

The element type "xsl:for-each" must be terminated by the matching end-tag "</xsl:for-each>".

- Caused due to either lack of end-tag for the specified tag or end-tags placed in the wrong order.
Answers

Exercise 1

Use NetBeans to create a markup file describing a restaurant. The markup should identify the name and address of the restaurant and the type of food or foods in which it specializes.

Answer

Submitted by Princess Anne Rayborn

```xml
<restaurant>
  <restaurantname>My favorite restaurant is <restaurantname>Olive Garden</restaurantname>. It is a chain that has branches all over. I have been to the restaurants in <location>Houston</location>, <location>Marietta</location>, and <location>Alpharetta</location>. They serve <cuisine>Italian</cuisine> food – my favorite is the <dish>Stuffed Chicken Parmigiana</dish>, and I also recommend the <dish>tiramisu</dish>.
</restaurant>
```

Exercise 2

Let's assume we want to create a personnel file for a smaller company. What kind of data do we have? Analyse the data from following table. Our goal is to transform it into a markup file. Name of the company: 'Exercises inc.'

Answer

Submitted by Stefan Meyer (Team 1)

```xml
<company>
  <company_name>Exercises inc.</company_name>
  <department>finance and accounting</department>
  <person>
    <firstname>Tobias</firstname>
    <lastname>Boeswald</lastname>
    <street>Laxenburger str. 384</street>
    <city>Vienna</city>
    <country>Austria</country>
    <date_of_birth>02/07/1974</date_of_birth>
  </person>
</company>
```
Exercise 1

1. Create an XML schema to describe the most popular restaurant in a city. Use either the all order indicator or the choice order indicator in your schema. Check that it is well-formed and valid.

2. Using the schema, create an XML document and populate it with data for one city that has a most popular restaurant as well as two or more other restaurants. Check that it is well-formed and valid.

3. Write an XML style sheet to display the most popular restaurants sorted by name if the restaurant is located in a city with population > 5,000.

**XML schema**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xml version="1.0" encoding="UTF-8"?>
<!--
    Document : city_restaurant.xsd
    Created on: February 17, 2004, 5:55 PM
    Author: Shirley Loh
    Description:
    Purpose of XML Schema document follows.
  -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <!--
    Directory
  -->
  <xsd:element name="directory">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="city" type="cityDetails" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--
    City
  -->
  <xsd:complexType name="cityDetails">
    <xsd:sequence>
      <xsd:element name="cityName" type="xsd:string"/>
      <xsd:element name="adminUnit" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="population" type="xsd:integer"/>
      <xsd:element name="area" type="xsd:integer"/>
      <xsd:element name="elevation" type="xsd:integer"/>
      <xsd:element name="longitude" type="xsd:decimal"/>
      <xsd:element name="latitude" type="xsd:decimal"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="history" type="xsd:string"/>
      <xsd:element name="topRestaurant" type="restaurantDetails" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="restaurant" type="restaurantDetails" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <!--
    Restaurant
  -->
  <xsd:complexType name="restaurantDetails">
    <xsd:all>
      <xsd:element name="restaurantName" type="xsd:string"/>
      <xsd:element name="streetAddress" type="xsd:string"/>
      <xsd:element name="phone" type="xsd:string"/>
      <xsd:element name="fax" type="xsd:string" minOccurs="0" maxOccurs="0"/>
      <xsd:element name="websiteURL" type="xsd:anyURI" minOccurs="0" maxOccurs="0"/>
      <xsd:element name="capacity" type="xsd:integer" minOccurs="0" maxOccurs="0"/>
      <xsd:element name="cuisine" type="xsd:string"/>
      <xsd:element name="operatingHour" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>
```

**XML document**

```xml
<department name="finance and accounting"/>
<person>
  <firstname>Dimitri</firstname>
  <lastname>Felber</lastname>
  <street>Neuburger str. 19a</street>
  <city>Passau</city>
  <country>Germany</country>
  <date_of_birth>05/12/1967</date_of_birth>
  <phone_nr>00498510/523456</phone_nr>
  <department name="finance and accounting"/>
</person>

<department name="human resources"/>
<person>
  <firstname>Stefan</firstname>
  <lastname>Meyer</lastname>
  <street>Breite Gasse 10</street>
  <city>Nuremberg</city>
  <country>Germany</country>
  <phone_nr>00499110/45365</phone_nr>
  <department name="human resources"/>
</person>
```
<directory xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='city_restaurant.xsd'>
  <city>
    <cityName>Belmopan</cityName>
    <adminUnit>Cayo</adminUnit>
    <country>Belize</country>
    <population>11100</population>
    <area>5</area>
    <elevation>130</elevation>
    <longitude>12.3</longitude>
    <latitude>123.4</latitude>
    <description>Belmopan is the capital of Belize</description>
    <history>Belmopan was established following devastation of the former capital, Belize City, by Hurricane Hattie in 1965. High ground and open space influenced the choice and ground-breaking began in 1966. By 1970 most government offices and operations had already moved to the new location.</history>
  </city>
  <restaurant>
    <restaurantName>The Caladium Restaurant</restaurantName>
    <streetAddress>Market Square</streetAddress>
    <phone>501-08-22754</phone>
    <cuisine>Belizean and International</cuisine>
    <operatingHour>7:00 a.m. to 8:00 p.m. Mon thru Sat</operatingHour>
  </restaurant>
  <restaurant>
    <restaurantName>Bull Frog Restaurant</restaurantName>
    <streetAddress>25 Half Moon Avenue</streetAddress>
    <phone>501-822-3425</phone>
    <capacity>90</capacity>
    <cuisine>Belizean</cuisine>
    <operatingHour>7:00 a.m. to 9:30 p.m. Mon thru Sun</operatingHour>
  </restaurant>

  <city>
    <cityName>Kuala Lumpur</cityName>
    <adminUnit>Selangor</adminUnit>
    <country>Malaysia</country>
    <population>1448600</population>
    <area>243</area>
    <elevation>111</elevation>
    <longitude>101.71</longitude>
    <latitude>3.16</latitude>
    <description>Kuala Lumpur is the capital of Malaysia and is the largest city in the nation.</description>
    <history>The city was founded in 1857 by Chinese tin miners and superseded Klang. In 1888 the British government transferred their headquarters from Klang to Kuala Lumpur, and in 1896 it became the capital of Malaysia.</history>
  </city>
  <topRestaurant>
    <restaurantName>EEST</restaurantName>
    <streetAddress>Level 1, 199 Jalan Bukit Bintang</streetAddress>
    <phone>011-603-2731-8333</phone>
    <fax>011-603-2731-8000</fax>
    <websiteURL>http://www.alloexpat.com/westin_kuala lumpur_malaysia.htm</websiteURL>
    <cuisine>Japanese and South East Asian</cuisine>
    <operatingHour>11:00 a.m. to 11:30 p.m. Mon thru Sun</operatingHour>
  </topRestaurant>
  <restaurant>
    <restaurantName>1957 Bistro</restaurantName>
    <streetAddress>22, Jalan 25/70A Desa Sri Hartamas</streetAddress>
    <phone>011-603-430-1030</phone>
    <cuisine>Fusion</cuisine>
    <operatingHour>9:00 a.m. to 11:00 p.m. Mon thru Sun</operatingHour>
  </restaurant>
  <restaurant>
    <restaurantName>Herbal Soup House</restaurantName>
    <streetAddress>19 Jalan Telawi Dua</streetAddress>
    <phone>011-603-998-1232</phone>
    <capacity>180</capacity>
    <cuisine>Asian</cuisine>
    <operatingHour>10:00 a.m. to 10:30 p.m. Mon thru Sun</operatingHour>
  </restaurant>
</city>
</directory>
Exercise 2

1. Create an XML schema to represent the most popular professors at a university. Include first name, last name, department, and years teaching.
2. Using the XML schema, create an XML document and populate it with data about a university and its most popular professor. Use at least three universities and two professors at each university.
3. Write an XML style sheet to display the most popular professors at each school sorted by the professor names.

**XML schema**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xsd:element name="universityGuide">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="university" type="universityDetails" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="universityDetails">
    <xsd:sequence>
      <xsd:element name="universityName" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="population" type="xsd:integer" default="0"/>
      <xsd:element name="area" type="xsd:integer"/>
      <xsd:element name="undergradNumber" type="xsd:integer"/>
      <xsd:element name="descriptionUniv" type="xsd:string"/>
      <xsd:element name="topProf" type="professorDetails" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="professor" type="professorDetails" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="professorDetails">
    <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string"/>
      <xsd:element name="lastName" type="xsd:string"/>
      <xsd:element name="yearsTeaching" type="xsd:integer"/>
      <xsd:element name="streetAddress" type="xsd:string"/>
      <xsd:element name="postalCode" type="xsd:string"/>
      <xsd:element name="telephoneNumber" type="xsd:string"/>
      <xsd:element name="professorRating" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

**XML document**

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <university>
    <universityName>University of Georgia</universityName>
    <city>Athens</city>
    <state>GA</state>
    <population>32000</population>
    <area>3654</area>
    <undergradNumber>22000</undergradNumber>
    <descriptionUniv>Located in Athens, GA. This campus is home to over 32,000 students. Majors range from Biology to Management.</descriptionUniv>
    <topProf>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <yearsTeaching>10</yearsTeaching>
      <streetAddress>100 University Drive</streetAddress>
      <postalCode>30607</postalCode>
      <telephoneNumber>555-1234</telephoneNumber>
      <professorRating>4</professorRating>
    </topProf>
    <professor>
      <firstName>Jane</firstName>
      <lastName>Smith</lastName>
      <yearsTeaching>5</yearsTeaching>
      <streetAddress>200 University Road</streetAddress>
      <postalCode>30608</postalCode>
      <telephoneNumber>555-5678</telephoneNumber>
      <professorRating>3</professorRating>
    </professor>
  </university>
</universityGuide>
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>OneToOneXMLStylesheet.xsl - Universities and Professors</title>
      </head>
      <body>
        <xsl:choose>
          <xsl:when test="count(/university) = 1">
            <xsl:for-each select="/university">
              <h1><xsl:value-of select="universityName"/></h1>
              <p><strong>City:</strong> <xsl:value-of select="city"/></p>
              <p><strong>State:</strong> <xsl:value-of select="state"/></p>
              <p><strong>Population:</strong> <xsl:value-of select="population"/></p>
              <p><strong>Area:</strong> <xsl:value-of select="area"/></p>
              <p><strong>Undergrad Number:</strong> <xsl:value-of select="undergradNumber"/></p>
              <p><strong>Description:</strong> <xsl:value-of select="descriptionUniv"/></p>
            </xsl:for-each>
          </xsl:when>
          <xsl:otherwise>
            <xsl:for-each select="/university">
              <h1><xsl:value-of select="universityName"/></h1>
              <p><strong>City:</strong> <xsl:value-of select="city"/></p>
              <p><strong>State:</strong> <xsl:value-of select="state"/></p>
              <p><strong>Population:</strong> <xsl:value-of select="population"/></p>
              <p><strong>Area:</strong> <xsl:value-of select="area"/></p>
              <p><strong>Undergrad Number:</strong> <xsl:value-of select="undergradNumber"/></p>
              <p><strong>Description:</strong> <xsl:value-of select="descriptionUniv"/></p>
            </xsl:for-each>
          </xsl:otherwise>
        </xsl:choose>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
Exercises 2a

- Create a schema describing one team in a relay race. The information needs to describe the runners in the race as well as from whom the runner received the baton. Create an XML file containing sample data for this schema and populate it with at least four runners. Make sure the XML document is well-formed and valid.

Answer:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<race xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="relayXSD.xsd">

<runner primaryKey="1">
    <firstName>Jesse</firstName>
    <lastName>Owens</lastName>
</runner>

<runner primaryKey="2">
    <firstName>Bruce</firstName>
    <lastName>Jenner</lastName>
    <predecessor foreignKey="1"/>
</runner>

<runner primaryKey="3">
    <firstName>Clarke</firstName>
    <lastName>Kent</lastName>
    <predecessor foreignKey="2"/>
</runner>

<runner primaryKey="4">
    <firstName>Steve</firstName>
    <lastName>Austin</lastName>
    <predecessor foreignKey="3"/>
</runner>

<runner primaryKey="5">
    <firstName>Jamie</firstName>
    <lastName>Summers</lastName>
    <predecessor foreignKey="4"/>
</runner>

<runner primaryKey="6">
    <firstName>Road</firstName>
    <predecessor foreignKey="5"/>
</runner>

</race>
```
File relayXML.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml.dtd">

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xsd:element name="race">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="runner" type="runnerType" maxOccurs="10" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:keyref name="predecessorKey" refer="runKey">
    <xsd:selector xpath="runner/predecessor" />
    <xsd:field xpath="@foreignKey" />
  </xsd:keyref>

  <xsd:unique name="runIdChecker">
    <xsd:selector xpath="runner" />
    <xsd:field xpath="@primaryKey" />
  </xsd:unique>

  <xsd:unique name="oneToOneChecker">
    <xsd:selector xpath="runner/predecessor" />
    <xsd:field xpath="@foreignKey" />
  </xsd:unique>

  <xsd:key name="runKey">
    <xsd:selector xpath="runner" />
    <xsd:field xpath="@primaryKey" />
  </xsd:key>

  <xsd:complexType name="runnerType">
    <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string" />
      <xsd:element name="lastName" type="xsd:string" />
      <xsd:element name="predecessor" type="predecessorType" minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="primaryKey" type="xsd:long" use="required" />
  </xsd:complexType>

  <xsd:complexType name="predecessorType">
    <xsd:attribute name="foreignKey" type="xsd:long" use="required" />
  </xsd:complexType>
</xsd:schema>
```

File relayXSD.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml.dtd">

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xsd:element name="race">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="runner" type="runnerType" maxOccurs="10" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:keyref name="predecessorKey" refer="runKey">
    <xsd:selector xpath="runner/predecessor" />
    <xsd:field xpath="@foreignKey" />
  </xsd:keyref>

  <xsd:unique name="runIdChecker">
    <xsd:selector xpath="runner" />
    <xsd:field xpath="@primaryKey" />
  </xsd:unique>

  <xsd:unique name="oneToOneChecker">
    <xsd:selector xpath="runner/predecessor" />
    <xsd:field xpath="@foreignKey" />
  </xsd:unique>

  <xsd:key name="runKey">
    <xsd:selector xpath="runner" />
    <xsd:field xpath="@primaryKey" />
  </xsd:key>

  <xsd:complexType name="runnerType">
    <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string" />
      <xsd:element name="lastName" type="xsd:string" />
      <xsd:element name="predecessor" type="predecessorType" minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="primaryKey" type="xsd:long" use="required" />
  </xsd:complexType>

  <xsd:complexType name="predecessorType">
    <xsd:attribute name="foreignKey" type="xsd:long" use="required" />
  </xsd:complexType>
</xsd:schema>
```

File appetizerXML.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stylesheet PUBLIC "-//W3C//DTD XSL-FO 1.0//EN" "http://www.w3.org/1999/XSL/Format">

  <html>
    <head>
      <style>
        BODY {background-color:darkblue}
      </style>
    </head>
    <body>
      <table style="background-color:gray; color:darkblue;margin-left:auto; margin-right:auto">
        <tr style="background-color:lightblue;font-weight:bold">
          <td>Runner First Name</td>
          <td>Runner Last Name</td>
          <td>Predecessor First Name</td>
          <td>Predecessor Last Name</td>
        </tr>
        <xsl:for-each select="//runner">
          <tr style="background-color:salmon">
            <td><xsl:value-of select="firstName" /></td>
            <td><xsl:value-of select="lastName" /></td>
            <td><xsl:value-of select="predecessor/firstName" /></td>
            <td><xsl:value-of select="predecessor/lastName" /></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:stylesheet>
```
Each state has many life insurance sales people, and each region (North, South, East, West) of the state has one regional head salesperson. A salesperson can only sell in one region and can only have one head salesperson he/she is working under. Create a schema to describe this relationship. Create an XML document and populate it with data for two states. Make sure the XML document is well formed and valid. Each state should have data for at least two regions, and each region should have at least three salesmen, including the region head. Each salesperson should be uniquely identified by a number. Each state should be uniquely identified by its two letter abbreviation. Create an XSL stylesheet to display this data, with a separate table for each state. If the salesperson is a regional head, it should say so. Create an entity named doublespace in the stylesheet that is the equivalent to two carriage returns. Use the new entity to create two blank lines between the two tables.

Answer:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<insuranceInfo xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='file:/Users/russell/Courses/MIST7700/XML/ch6-assn/insuranceXSD.xsd'>
  <state stateAbbr = "GA"> Georgia
    <region regionId = "1"> North
      <salesPerson salesPersonId="1"> Joe
        <firstName>Joe</firstName>
        <lastName>Schmo</lastName>
      </salesPerson>
      <salesPerson salesPersonId="2"> Jim
        <firstName>Jim</firstName>
        <lastName>Beam</lastName>
        <regionHead fk_salesPersonId = "1"/>
      </salesPerson>
      <salesPerson salesPersonId="3"> Johnny
        <firstName>Johnny</firstName>
        <lastName>Walker</lastName>
        <regionHead fk_salesPersonId = "1"/>
      </salesPerson>
    </region>
    <region regionId="2"> South
      <salesPerson salesPersonId="4"> Ian
        <firstName>Ian</firstName>
        <lastName>Fleming</lastName>
      </salesPerson>
      <salesPerson salesPersonId="5"> Tom
        <firstName>Tom</firstName>
        <lastName>Clancy</lastName>
        <regionHead fk_salesPersonId = "4"/>
      </salesPerson>
      <salesPerson salesPersonId="6"> Conan
        <firstName>Conan</firstName>
        <lastName>Doyle</lastName>
        <regionHead fk_salesPersonId = "4"/>
      </salesPerson>
    </region>
  </state>
  <state stateAbbr = "CA"> California
    <region regionId = "1"> West
      <salesPerson salesPersonId="7"> Conan
        <firstName>Conan</firstName>
        <lastName>Doyle</lastName>
        <regionHead fk_salesPersonId = "4"/>
      </salesPerson>
      <salesPerson salesPersonId="8"> Tom
        <firstName>Tom</firstName>
        <lastName>Clancy</lastName>
        <regionHead fk_salesPersonId = "4"/>
      </salesPerson>
      <salesPerson salesPersonId="9"> Ian
        <firstName>Ian</firstName>
        <lastName>Fleming</lastName>
      </salesPerson>
    </region>
    <region regionId="2"> East
      <salesPerson salesPersonId="10"> Johnny
        <firstName>Johnny</firstName>
        <lastName>Walker</lastName>
        <regionHead fk_salesPersonId = "1"/>
      </salesPerson>
      <salesPerson salesPersonId="11"> Joe
        <firstName>Joe</firstName>
        <lastName>Schmo</lastName>
      </salesPerson>
      <salesPerson salesPersonId="12"> Jim
        <firstName>Jim</firstName>
        <lastName>Beam</lastName>
        <regionHead fk_salesPersonId = "1"/>
      </salesPerson>
    </region>
  </state>
</insuranceInfo>
```
File insuranceXML.xml

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

&lt;!--
  Document : appetizerXML.xml
  Created on : March 28, 2004, 9:46 PM
  Author : Jess Russell
  Description: Purpose of the document follows.
--&gt;
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="insuranceInfo">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="state" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="stateName" type="xsd:string"/>
              <xsd:element name="region" type="regionInfo" maxOccurs="4"/>
            </xsd:sequence>
            <xsd:attribute name="stateAbbr" type="xsd:string" use="required"/>
          </xsd:complexType>
          <xsd:unique name="regionChecker">
            <xsd:selector xpath="region"/>
            <xsd:field xpath="@regionId"/>
          </xsd:unique>
          <xsd:unique name="salesPersonChecker">
            <xsd:selector xpath="region/salesPerson"/>
            <xsd:field xpath="@salesPersonId"/>
          </xsd:unique>
          <xsd:key name="salesPersonKey">
            <xsd:selector xpath="salesPerson"/>
            <xsd:field xpath="@salesPersonId"/>
          </xsd:key>
          <xsd:keyref name="salesPersonForeignKey" refer="salesPersonKey">
            <xsd:selector xpath="state"/>
            <xsd:field xpath="@fk_salesPersonId"/>
          </xsd:keyref>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="stateChecker">
      <xsd:selector xpath="state"/>
      <xsd:field xpath="@stateAbbr"/>
    </xsd:unique>
  </xsd:element>
</xsd:schema>
A restaurant has many appetizers on its menu. A restaurant may also create a sampler dish which is a conglomeration of smaller portions of select appetizers. Create a schema to describe this relationship. Create an XML document and populate it with enough appetizers to create at least two sampler groups. Make sure the XML document is well formed and valid.

Answer:

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

<!-- The namespace attribute above is only necessary for XML parsers using the MSXML parser-->

<xs:schema

<xs:complexType name="regionInfo">
  <xs:sequence>
    <xs:element name="regionName" type="xsd:string"/>
    <xs:element name="salesPerson" type="salesPersonInfo" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="regionId" type="xsd:long" use="required"/>
</xs:complexType>

<xs:complexType name="salesPersonInfo">
  <xs:sequence>
    <xs:element name="firstName" type="xsd:string"/>
    <xs:element name="lastName" type="xsd:string"/>
    <xs:element name="regionHead" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:attribute name="fk_salesPersonId" type="xsd:long" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="salesPersonId" type="xsd:long" use="required"/>
</xs:complexType>

<xs:schema

<xs:complexType name="regionInfo">
  <xs:sequence>
    <xs:element name="regionName" type="xsd:string"/>
    <xs:element name="salesPerson" type="salesPersonInfo" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="regionId" type="xsd:long" use="required"/>
</xs:complexType>

<xs:complexType name="salesPersonInfo">
  <xs:sequence>
    <xs:element name="firstName" type="xsd:string"/>
    <xs:element name="lastName" type="xsd:string"/>
    <xs:element name="regionHead" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:attribute name="fk_salesPersonId" type="xsd:long" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="salesPersonId" type="xsd:long" use="required"/>
</xs:complexType>

<xs:schema

```
File appetizerXML.xml

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

<!--

Document : appetizerXML.xml
Created on : March 28, 2004, 9:46 PM
Author : Jess Russell
Description: Purpose of the document follows.

-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  
  <!--
  Document : appetizerXML.xml
  Created on : March 28, 2004, 9:46 PM
  Author : Jess Russell
  Description: Purpose of the document follows.
  
  -->

  <xsd:element name="appetizers">
    
    <xsd:complexType>
      
      <xsd:sequence>
        
        <xsd:element name="appetizer" type="appetizerInfo" minOccurs="0" maxOccurs="unbounded"/>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

  
</xsd:schema>
XML Exercises

In this exercise, you will modify the xsl-tree.xsl file we saw in chapter 8 XPath to...

1. Print the name attribute of every bigBranch element with thickness set as ‘thick’.
2. Print the name of the parent of each bigBranch.
3. Print the color of all leaves with a color attribute.

Answers

1. Print the name attribute of every bigBranch element with thickness set as ‘thick’

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/child::trunk">
    <html>
      <head>
        <title>XPath Tree Tests</title>
      </head>
      <body>
        <xsl:for-each select="child::bigBranch[attribute::thickness='thick']">
          <xsl:call-template name="print_out"/>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="print_out">
    <xsl:value-of select="attribute::name"/>
    <br/>
  </xsl:template>
</xsl:stylesheet>
```

Output

```
x
```

2. Print the name of the parent of each bigBranch

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/child::trunk">
    <html>
      <head>
        <title>XPath Tree Tests</title>
      </head>
      <body>
        <xsl:for-each select="child::bigBranch">
          <xsl:call-template name="print_out"/>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="print_out">
    <xsl:value-of select="parent::node()[attribute::name]"/>
    <br/>
  </xsl:template>
</xsl:stylesheet>
```
3. Print the color of all leaves with a color attribute

<xsl-tree3.xsl>

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/child::trunk">
        <html>
            <head>
                <title>XPath Tests</title>
            </head>
            <body>
                <xsl:for-each select="descendant::leaf[string(attribute::color) != '']">
                    <xsl:call-template name="print_out"/>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>
    <xsl:template name="print_out">
        <xsl:value-of select="attribute::name"/>
        ,
        <xsl:value-of select="attribute::color"/>
        <br/>
    </xsl:template>
</xsl:stylesheet>

Output

leaf1, brown
leaf5, purple
leaf9, black
leaf14, red
leaf17, red

Exercise 1

Here is one way you could do it:

To be able to hook up your external style sheet with your XML document you have to write:

<?xml-stylesheet type="text/css" href="exercise1.css"?>

at the start of the document, under

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

Like this:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?xml-stylesheet type="text/css" href="exercise1.css"?>
<inventory>
    <book>
        <isbn>91-1-300875-7</isbn>
        <title>Here you put in a title</title>
        <author>Here you put in a author</author>
        <publisher>&gyl;</publisher>
        <price>$30</price>
    </book>
    <book>
        <isbn>91-2-705229-X</isbn>
        <title>Title of second book</title>
        <author>Author of second book</author>
        <publisher>&sybex;</publisher>
        <price>$30</price>
    </book>
</inventory>

This java class file is associated with the movies.xml file used in the chapter. Place this java file and movies.xml in the same directory.

After compiling, this needs to be run from a command window and the format is:

> java SPParser movies.xml
Exercises

1. Create a window that contains two text boxes that are approximately 200 pixels down the window. Use either vertical or horizontal layout. Label each text box and set default values for each text box that are different from each other. Create a button labeled ‘Swap text’. Next, create a function that when the button is clicked, will swap the text from one text box to the other.

2. In this exercise, you will need to make two separate files. On one page, create a window that has two tabbed panels with different content on each. On the other page, simply make a popup menu with one menu item ‘Open’. Create a function that will redirect the browser to the tabbed page when the ‘Open’ menu option is clicked.
XulNum1.js

```javascript
function swap()
{
  var noDanaObj=document.getElementById('nodana');
  var onlyXulObj=document.getElementById('onlyxul');
  var noDanaStr=noDanaObj.getAttribute("value");
  var onlyXulStr=onlyXulObj.getAttribute("value");
  noDanaObj.setAttribute("value",onlyXulStr);
  onlyXulObj.setAttribute("value",noDanaStr);
}
```

XulNum1.xul

```xml
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="Ex1" title="Excercise 1" orient="vertical" xmlns= "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <script src="XulNum1.js" /> 
  <hbox width="200px" height="200px" flex = "1">
    <text value="There is no Dana..." id="nodana" style="font-size:18pt" /></hbox>
  <hbox width="200px" height="200px" flex = "1">
    <text value="There is only XUL" id="onlyxul" style="font-size:24pt" /></hbox>
  <box width="200px">
    <button id="swap" label="swap" default="true" oncommand="swap();" />
  </box>
</window>
```

### Answer

1. Change the HTML document below into an XHTML document conforming to the W3C's transitional standard. Validate your page using the validator available at [http://validator.w3.org/](http://validator.w3.org/).

#### Exercise 1

Below is an example of an invalid HTML document that contains a number of deprecated features. It is also badly structured. The document needs converting to XHTML with the content separated from the presentation.

**Exercise 1.a.**

a. Change the HTML document into a valid XHTML 1.0 Transitional document. Validate your page using the validator available at [http://validator.w3.org/](http://validator.w3.org/).

The XHTML 1.0 Transitional document:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "‐//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Convert HTML to XHTML</title>
  </head>
  <body text="blue">
    <h1>XHTML page</h1>
    <p><b>It is important for your site to be current with the most recent W3C standards.</b></p>
    <p><u>Welcome</u> to my <b>page</b>. I hope that you <i>enjoy</i> your stay.</p>
    <p><font color="#9900FF" face="Arial" size="+2">XHTML is similar to HTML</font></p>
    <a href="http://validator.w3.org">Validator</a>
  </body>
</html>
```

**Exercise 1.b.**

b. Starting with the model answer for question 1.a. identify all deprecated elements and attributes in the document and replace them with CSS in a linked external stylesheet. See [HyperText Markup Language/Tag List](https://developer.mozilla.org/en-US/docs/Web/HTML/Element) for details of deprecated elements.
The deprecated features are:
- the text attribute of the body element;
- the u element; and
- the font.

A possible stylesheet is:

```css
body {
    color: blue;
}
span#welcome {
    text-decoration: underline;
    /* It is generally a bad idea to underline anything other than hyperlinks. */
}
.highlight {
    color: #9900FF;
    font-size: 150%;
    font-family: Arial, sans-serif;
    /* It is good practice to include one as the CSS generic font families, e.g. sans-serif, as an alternative. */
}
```

The document with deprecated features removed – note the use of the id and class attributes to provide hooks for the CSS selectors.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "‐//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Convert HTML to XHTML</title>
<link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
<h1>XHTML page</h1>
<p><strong>It is important for your site to be current with the most recent W3C standards.</strong></p>
<p><span id="welcome">Welcome</span> to my <strong>page</strong>.</p>
<p>I hope that you <em>enjoy</em> your stay.</p>
<p class="highlight">XHTML is similar to HTML</p>
<p><a href="http://validator.w3.org">Validator</a></p>
</body>
</html>
```

**Exercise 1.c.**

c. Starting with the model answer for question 1.b. replace all presentational markup with semantic markup and ensure that all inline elements are contained in block-level elements. Change the DOCTYPE to XHTML 1.0 Strict and validate the page with the W3C Markup Validation Service.

The presentational elements are:
- b
- i
- br

The 'paragraph' beginning with a span element is a mixture of inline elements and text and so should be contained in a block-level element such as p (paragraph).

The a (anchor) element is inline and needs enclosing in a block-level element.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "‐//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Convert HTML to XHTML</title>
<link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
<h1>XHTML page</h1>
<p><strong>It is important for your site to be current with the most recent W3C standards.</strong></p>
<p><span id="welcome">Welcome</span> to my <strong>page</strong>.</p>
<p>I hope that you <em>enjoy</em> your stay.</p>
<p class="highlight">XHTML is similar to HTML</p>
<p><a href="http://validator.w3.org">Validator</a></p>
</body>
</html>
```

**SyncML Exercises**
2) Answer these questions: a. What is WBXML and why is it used? b. How do you foresee SyncML being used in the future? c. Name a problematic situation whereby SyncML is the best 'tool' for the job.

Answers

2a. WBXML is a form of XML that abbreviates XML tags by using a binary representation of the tag. Tags need to be abbreviated in order to work with mobile devices that have limited bandwidth and limited memory.

2b. SyncML could be used in any application in which synchronization is needed such as calendars, address books, phone books, inventory tracking, etc.

2c. An auto mechanic needs to view the work history on cars in the shop. A PDA can be used to view work logs, active work orders, and to log work done towards a current work order.

1. Create a simple SMIL file which displays the words, 'Hello World'. Confirm that the file works in a SMIL-conformant player.

2. Author a SMIL file which displays 'Hello World' for 3 seconds, then displays 'Goodbye World' for 1 second. Confirm that the file works in a SMIL-conformant player.

Answer

1. Submitted by Anne Rayborn Howard

File Exercise1.smil

```xml
<smil>
  <head>
    <layout>
      <root-layout width="200" height="75" background-color="white"/>
    </layout>
  </head>
  <body>
    <text src="Exercise1.rt" alt="Hello World" region="text" dur="1s" begin="0s"/>
  </body>
</smil>
```

File Exercise1.rt

```
Hello World!
```

2. Submitted by Anne Rayborn Howard

File Exercise2.smil

```xml
<smil>
  <head>
    <layout>
      <root-layout width="200" height="75" background-color="white"/>
    </layout>
  </head>
  <body>
    <text src="Exercise1.rt" alt="Hello World" region="text" dur="3s" begin="0s"/>
    <text src="Exercise2.rt" alt="Goodbye World" region="text" dur="1s" begin="0s"/>
  </body>
</smil>
```

File Exercise2.rt

```
Goodbye World!
```

3. Submitted by (your name here)

If you have MS Powerpoint, you can use the Smil Generator for Powerpoint presentations (http://www8.informatik.uni-erlangen.de/~aoklein/de/Softwareproject/SMILGenerator.htm#NO-302) to convert your powerpoint.

4. Submitted by (Your Name Here!)
Question

1. Create an RDF/XML document that describes an article of your choice (e.g. from magazines like CIO.com or ZDNet.com). Use the Dublin Core element set (http://dublincore.org/documents/dces/) and the Dublin Core terms (http://purl.org/dc/terms/) as a framework for your description. After completing the document please validate your work with the W3C RDF Validator (http://www.w3.org/RDF/Validator/).

Answer

```xml
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:dcterms="http://purl.org/dc/terms/">
   <rdf:Description
      rdf:about="http://www.cio.com/archive/101504/km.html">
     <dc:title>Less for Success</dc:title>
     <dc:creator>Alice Dragoon</dc:creator>
     <dc:subject>
       <rdf:Bag>
         <rdf:li>knowledge management</rdf:li>
         <rdf:li>technology investments</rdf:li>
       </rdf:Bag>
     </dc:subject>
     <dc:description>Forget the big bang approach. When it comes to demonstrating the value of knowledge management, a piecemeal strategy works best.</dc:description>
     <dc:publisher>CXO Media</dc:publisher>
     <dcterms:issued>October 15, 2004</dcterms:issued>
     <dc:format>text/html</dc:format>
     <dc:language>en</dc:language>
   </rdf:Description>
</rdf:RDF>
```

Topic Maps

1. Business Intelligence and XML (Answers)

Exercise 1

The end user wants to read the report in Japanese language, so you have to add a variable for Japanese language.

Answer

```xml
<variableList>
  <variable name="Variable1" type="locale">
    <expression>ReportLocale()</expression>
    <variableValueList>
      <variableValue value="fr"/>
      <variableValue value="ja"/>
    </variableValueList>
  </variable>
</variableList>
```


- This page was last modified on 12 November 2013, at 12:36.
- Text is available under the Creative Commons Attribution-ShareAlike License.; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy.