Book 4

From HTTP to XML and XSLT

U:\Book\Book_04.doc

Database Connectivity

Fehler! Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

Table of Contents

30 U:\Book\Book_04.doc

From HTTP to XML and XSLT

1 Foundations of HTTP and TCP/IP Protocols

TCP/IP is the de facto standard of nowadays networking. HTTP is the universal communication layer protocol. XML – the eXtensible Markup Language – has become the lingua franca of the internet. SOAP – the Simple Object Access protocol – is a document structure defined in XML with the purpose to transport object interfaces between distributed applications.

1.1 TCP/IP Network Protocol

The TCP over IP network protocol has become the universal standard for client server communication, such replacing other etablished network protocols as there has been the IBM NDIS protocol and the Novell IPX protocol.

IP stands for „Internet Protocol“

The same says it all and so you already know, where it comes from: it has been the original propriatary network protocol of the ancient CERN internet.

TCP stands for Transcient

The TCP protocol is one layer up and regroups IP packages in larger data streams.

Server listen to ports

All TCP/IP communication is based on the IP port. A physical server – the machine – has a unique IP address attached such as 127.0.0.1 or 169.128.1.1 . In addition to that, every machines supplies up to 65235 IP ports, represented by a sixteen digit bit string. An IP compatible server software will then listen to one port. When a message is sent to the specific IP addresses' Ipport, the software listening to port will take the data stream and react on it to ist discretion.

Listeners are servers

A server program attached to an IP port is called a listener because it listens to the data traffic arriving at that port.

Figure 1: Typical port assignment in an IP environment

FTP	21
TELNET	23
HTTP	80
SAP R/3	1023
LPD or SAPLPD	

⇨

1.2 HTTP Communication With A Webserver

An HTTP server and a browser are typical client-server application with the browser as the client and the HTTP server as the server. Both establish a simple bi-directional communication using a number of task specific protocols on top of the TCP/IP layer.

HTTP is a human readable ASCII stream

HTTP is a plain ASCII protocol. Bits are grouped as tupelos of 8 bits and the resulting bytes are interpreted as alphanumeric characters according the ASCII encoding scheme. This makes the protocol human readable and also understandable by every modern computer be it a mainframe, a PC or a handheld device.

HTTP is communication between a requester (client) and a responder

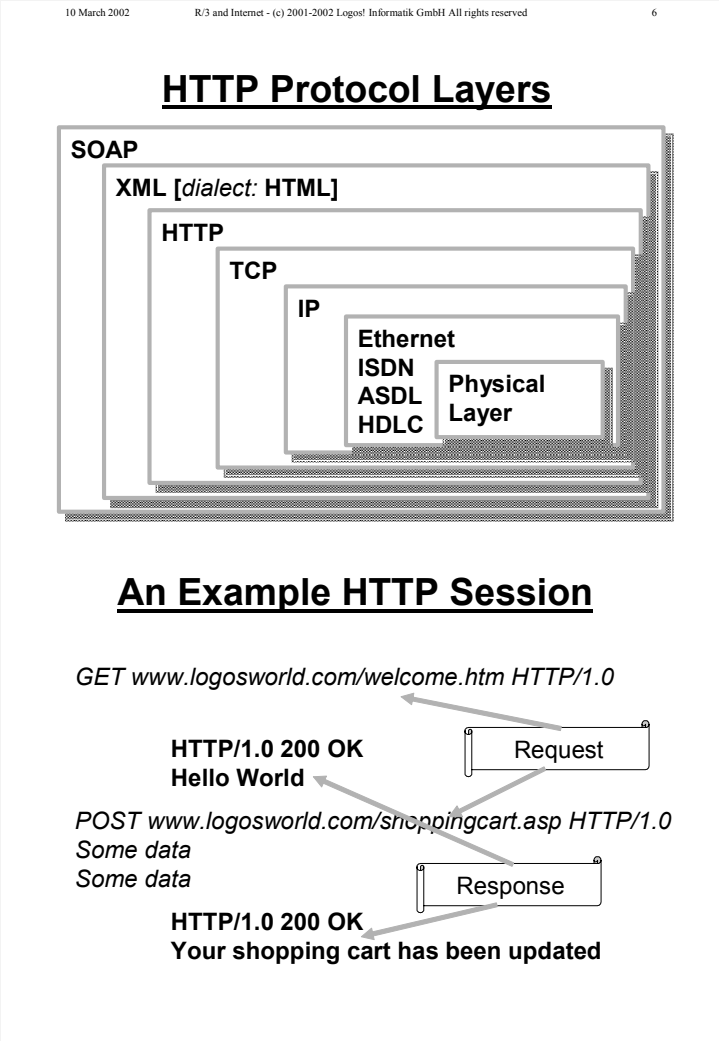
HTTP communication is a dialogue between a browser client and a web server. The browser client sends requests to the server, which responds to the them.

Two types of requests: GET and POST

HTTP knows two principle kinds of requests: the GET request and the POST request. A GET requests sends a string of information (the URL) to the web server and expects an appropriate response, in other words: it asks to GET back an HTML page. The POST request, does principally the same as a GET request, but

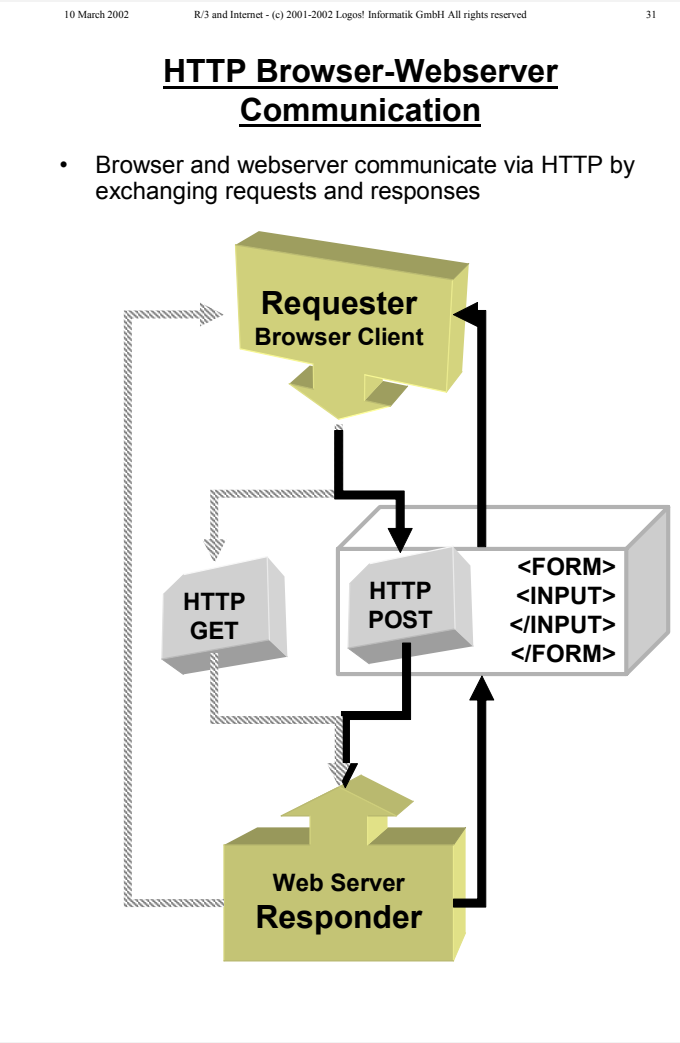
65 a POST is used to send additional data along with the URL. This data – called FORM data – is packed in the body of the request and is usually sent to be handled by the web server. In HTML the data sent with a POST is everything found between the <INPUT> …. </INPUT> tags of an HTML <FORM>.

Table 1: Hierarchy of protocol layers



⇨

Table 2: Communication via HTTP between browser and webserver



⇨

1.3 Technical Details of An HTTP Session

Communication is always started by the client

Communication is always started by the client, i.e. the browser. The browser initially sends a request to the web server, which interprets the request and sends back a response using the required protocol. A request from the browser to the web server is a simple ASCII command string that may look as follows:

```
GET postinfo.htm HTTP/1.0
{ Leave a blank line}
```

You can test the HTTP communication with telnet

If you like you can experiment with this communication easily by using TELNET or any other dumb terminal emulator. Execute TELNET from a DOS prompt as follows:

```
telnet localhost 80
```

This assumes that you run telnet on the same machine where you have installed the web server. The 80 is the TCP/IP port used by the web server. The commonly used ports are 80 or 8080 depending on your installation, but the number may be arbitrarily set by your web server administrator.

When you contact the web server successfully (usually your TELNET screen will be cleared and the TELNET session falls into the listen mode if everything is OK, otherwise you see an error message). After you have successfully contacted the server, enter the HTTP request command:

GET postinfo.htm HTTP/1.0

{ Leave a blank line}

The blank line is the indicator for the web server that the request is finished and the client waits for a response. (If a blank line seems weird to you, remember that the web server simply sees to the subsequent CR/LF, i.e. line break, which is the common file termination sequence in UNIX). The example assumes that the web server has the file postinfo.htm in its base directory. To display any other file specify the fully qualified path, e.g. for http://localhost/asp/helloworld.htm you would form the request:

GET /asp/helloworld.htm HTTP/1.0

Leave a blank line

As a response the TELNET screen will display the response of the web server, i.e. the content of the file postinfo.htm.

If you have not yet set up your web server, you may try the following:

telnet www.yahoo.com 80

GET index.htm HTTP/1.0

{ Leave a blank line}

If you connect to a the internet through a proxy server, then you have to TELNET the proxy server instead and specify the full URL then, e.g. like the following.

telnet myproxy 80

GET http://www.yahoo.com/index.htm HTTP/1.0

{ Leave a blank line}

As a result of the request you will receive a string of data with a well formed header line like the following:

HTTP/1.0 200 OK

The remainder of the message is the resulting HTML data.

In case of an error you will receive a header, indicating that there is a problem.

HTTP/1.0 404 NOT FOUND

The remainder in such cases may be an explanatory message detailing the kind of problem that has occurred.

1.4 Typical Protocols

There are many protocols for different purposes

The example above used the HTTP protocol, the standard protocol for the world wide web to exchange HTML documents. HTTP is one of many protocols, that is

why you would specify the protocol along with the URL – Universal Resource Locator – in order to indicate which protocol should be used.

http: HyperText Transfer Protocol	Tells the server that the requester (i.e. the browser) wants to receive proper HTML coded documents.
https: HyperText Transfer Protocol Secure	This is the same as HTTP, but all data is exchanged using the SSL encryption method, so that data remains readable for the requestor only.
ftp: File Transfer Protocol	FTP is mainly used to exchange files between server and client, i.e. the requested files are transferred without adding any additional information or formatting strings.
References:	A good tutorial on HTTP is found on http://www.jmarshall.com/easy/http/

2 XML – Extended Markup Language

XML is a plain ASCII document format meant to exchange data between correspondents. In order to create large distributed projects you should sooner or later become familiar with XML.

2.1 XML As The Lingua Franca Of The Internet

XML is the lingua franca of the internet

XML is known as the lingua franca of the internet. It is a plain text language, like HTML, which is structured but also human readable at the same time. A structured language is a precondition for efficient and deterministic automated processing, while the human readability makes error processing much easier when the automated processing failed for some reason.

XML Farm

This is a simple example of how to define a two dimensional database table in XML.

A table like this

Name	Weight	Gender
Cow	420	F
Pig	120	M

could be declared as XML as follows:

Listing 1: Example of a simple XML farm

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns="x-schema:U:\examples\XML\farmschema.xml">
 <Animal>
      <Name>Cow</Name>
      <Weight>420</Weight>
      <Gender>F</Gender>
 </Animal>
 <Animal>
      <Name>Pig</Name>
      <Weight>120</Weight>
      <Gender>M</Gender>
 </Animal>
</Farm>
```

⇨

What happened here?

In a traditional IT environment you might have created a file similar to the following:

```
Cow     000420F
Pig     000120M
```

The problem with a file like this obvious: if you receive it without any further hint, it is pretty useless, because you neither know where the columns begin and start nor how they are meant to be interpreted. Therefore we wrap in XML every field in an opening and a closing tag, that tells us the field's name.

```
<Name>Cow</Name><Weight>420</Weight><Gender>F</Gender>
<Name>Pig</Name><Weight>120</Weight><Gender>M</Gender>
```

When sending data through an electronic media, you may face some restriction. E.g. in HTTP you may not put carriage return <CR> and line feed <LF> characters arbitrarily into your data stream. E.g. two consecutive <CR><LF> in an HTTP screen signal a section break of the protocol, usually it tells the end of the transmission package. In addition, the line length may be limited by the transmission protocol (which is not the case in HTTP). To allow unlimited line sizes for the transmission file, the line is delimited by a row indicator, grouping the fields into rows of fields.

```
 <Animal><Name>Cow</Name><Weight>420</Weight><Gender>F</Gender></An
imal>
 <Animal><Name>Pig</Name><Weight>120</Weight><Gender>M</Gender></An
imal>
```

To put the data into a calculable formal structure, the whole data package is wrapped into a single envelope, that we call <FARM>.

```
<Farm>
 <Animal><Name>Cow</Name><Weight>420</Weight><Gender>F</Gender></An
imal>
 <Animal><Name>Pig</Name><Weight>120</Weight><Gender>M</Gender></An
imal>
</Farm>
```

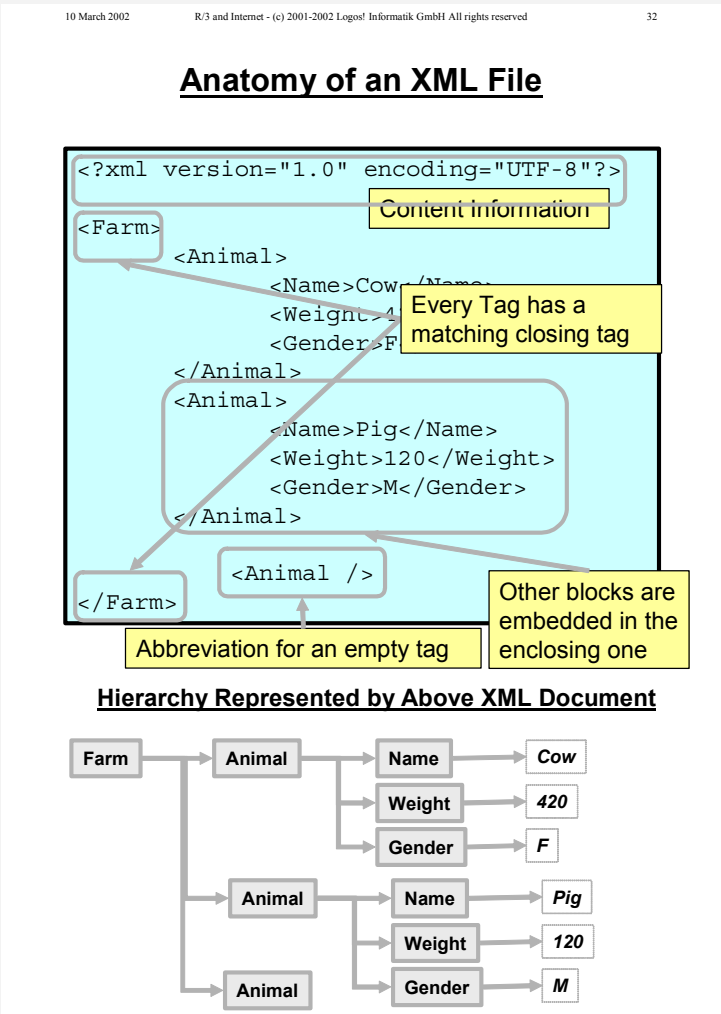
Finally we add some administrative information. The <?xml> directive is a harmonised statement for the receiver of the XML stream to tell something about the content of the XML stream, in our case it tells us the XML version expected to understand the following data and the character encoding used (options are e.g. UTF-8, ANSI, WINDOWS)

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

The additional attribute used with the envelope tag <Farm> tells a reference to a formal definition, that can be used to verify the content and like in our case which namespace is used.

```
<Farm xmlns="x-schema:U:\examples\XML\farmschema.xml">
```

Table 3: Anatomy of an XML File



⇨

2.2 The Microsoft XML Parser

Microsoft supplies together with the later versions of the Internet Explorer a powerful XML class, which can be used by everyone to parse and build XML documents.

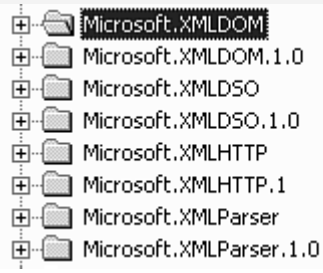
XML Document Browser e.g. XMLSPY

You can browse XML documents conveniently with Internet Explorer 5 or greater and Netscape 6 or greater. However, the formatting capabilities of the browsers are limited and editing documents is not supported. Therefore you should look out for a good XML editor that is able to display structured documents in matrix formats, like XMLspy found at http://www.xmlspy.com/ which is a complete XML IDE development environment.

XML Document ActiveX – Microsoft.XMLDOM

Microsoft provides a complete XML building and parsing library (msxml.dll). This class can create a valid XML document and if assigned a document, the document is parsed and the components are presented in a tree like structure as children of the document object. That makes programming easy.

Figure 2: Registry entries of XML related DLLs



⇨

Figure 3: Details of the Microsoft.XMLDOM class

Microsoft.XMLDOM.1.0	
ProgID:	Microsoft.XMLDOM.1.0
Version:	1.0
CLSID:	{2933BF90-7B36-11d2-B20E-00C04F983E60}
TypeLib Name:	Not Defined
TypeLib:	{d63e0ce2-a0a2-11d0-9c02-00c04fc99c8e}
File Location:	D:\WINNT\System32\msxml.dll
Status:	File Exists
File Size:	537360 Bytes
Created:	11/18/00 21:28:11
Last Modified:	12.10.1999 13:00
Accessed:	04/30/01
File Description:	XML OM für Win32
Company Name:	Microsoft Corporation
File Version:	5.00.2920.0000
Internal Name:	MsXML.dll
Legal Copyright:	Copyright (C) Microsoft Corp. 1981-1999
Original Filename:	MsXML.dll
Product Name:	Betriebssystem Microsoft(R) Windows (R) 2000
Product Version:	5.00.2920.0000
VB Object Creation	Set obj = CreateObject("Microsoft.XMLDOM.1")
VB Object Type	Dim obj As New XMLDocument

⇨

2.3 XML – Schemas

Schema creation

The next major step is to create a schema which is a formal way of defining and validating the content of an XML document. (A well-formed XML document that conforms to its schema is said to be valid.)

The schema is how we assign the data types to each tag and any attributes that are contained in the XML document. A schema is a structured document which must obey XML syntax rules. It is composed of a series of predefined tags and attributes that are part of the XML language and are used to set the data types for the values associated with our custom tags. Simply put, not only do we get to create custom XML tags, but we can also denote that an XML data value is, for example, an integer data type. This ability to assign specific data types to specific XML data values is one of the reasons why XML has attracted so much attention.

DCD: Document Content Description

A DTD is another XML block that describes the content of the following data.

210

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <ElementType name="Name" content="textOnly" dt:type="string"/>
 <ElementType name="Family" content="textOnly" dt:type="string"/>
 <ElementType name="Weight" content="textOnly" dt:type="int"/>
 <ElementType name="Gender" content="textOnly" dt:type="string"/>

 <ElementType name="Animal" content="mixed">
      <element type="Name" minOccurs="1" maxOccurs="*"/>
      <element type="Family" minOccurs="1" maxOccurs="*"/>
      <element type="Weight" minOccurs="1" maxOccurs="*"/>
      <element type="Gender" minOccurs="1" maxOccurs="*"/>
 </ElementType>

 <ElementType name="Farm" content="mixed">
      <element type="Animal" minOccurs="1" maxOccurs="*"/>
 </ElementType>
</Schema>
```

XML Schema

An XML Schema for the animal farm may look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN" ""
[
 <!ENTITY % p 'xsd:'>
 <!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <xsd:complexType name="TyFamily" content="elementOnly">
      <xsd:sequence>
           <xsd:element name="TyName">
                <xsd:simpleType base="xsd:string">
                     <xsd:enumeration value="Cow"/>
                     <xsd:enumeration value="Pig"/>
                </xsd:simpleType>
           </xsd:element>
      </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="TyGender" content="elementOnly">
      <xsd:sequence>
           <xsd:element name="TyGender">
                <xsd:simpleType base="xsd:string">
                     <xsd:enumeration value="F"/>
                     <xsd:enumeration value="M"/>
                </xsd:simpleType>
           </xsd:element>
      </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="AnimalType" content="elementOnly">
      <xsd:sequence>
           <xsd:element name="Family" type="TyFamily"/>
           <xsd:element name="Weight" type="xsd:short"/>
           <xsd:element name="Gender" type="TyGender"/>
      </xsd:sequence>
 </xsd:complexType>

<xsd:element name="Farm">
 <xsd:complexType content="elementOnly">
      <xsd:sequence>
           <xsd:element name="Animal" type="AnimalType"
                minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="xmlns:xsi"
           type="xsd:uriReference" use="default"
           value="http://www.w3.org/1999/XMLSchema-instance"/>
      <xsd:attribute name="xsi:noNamespaceSchemaLocation"
                     type="xsd:string"/>
      <xsd:attribute name="xsi:schemaLocation"
                     type="xsd:string"/>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

⇨

215 You may find it helpful to see, that a schema is more or less analogous to the type declaration section of any modern programming language, like as in the table below. However, the XML schemas allow a much more complex attribute setting to any type.

ABAP IV	Visual Basic	XML Schema
Types: TyFamily Type C(12)	Public Enum TyFamily Cow = 1 Pig = 2 End Enum	
Types:	Public Enum TyGender	

TyGender Type C(1)	male = 1 female = 2 End Enum	
Types: Begin Of Animal Name(32), Family Type TyFamily, Weight Type P, Gender Type TyGender, End Of Animal.	Public Type Animal Name As String Family As TyFamily Weight As Long Gender As TyGender End Type	<ElementType name="Animal" content="mixed"> <element type="Name"/> <element type="Family"/> <element type="Weight"/> <element type="Gender"/> </ElementType>

⇨

A schema can be part of the XML document or a separate file

A schema can be part of the XML document or a separate file. For our examples, we will create a separate schema file to allow you to view the resulting document.

Fortunately, if you can write HTML code, you can write a schema document. Here are the XML tags and attributes that we will use to create a schema:

2.4 XML Tags Used In Schemas

The following is a short overview of useful tags used in a DTD schema, that we found us using frequently. They are by far not complete. However, writing XML schemes is like formatting in type setting: although you have the options to use an unlimited number of different tags and formatting, it is wise to use as few as possible to allow legibility of the programs.

Schema

The **Schema** tag serves as a container element that delimits the beginning and end of the schema. This tag must be closed and please note the exact spelling with regard to case.

Xmlns

The **xmlns** attribute is used to declare the schema XML namespace. The value is a URL or URN address that the browser will access to get information on how to parse and validate the code.

xmlns:dt

The **xmlns:dt** attribute is used to declare the data types of the schema XML namespace. The value is a URL or URN address that the browser will access to get information on the data types to allow code validation.

If you are using IE 5 to view your XML document, then you must include the **xmlns** and the **xmlns:dt** attributes exactly as displayed below:

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

AttributeType

The **AttributeType** tag is used to declare the data type for an attribute of an XML tag. This tag must be closed and please note the exact spelling with regard to case.

name

The **name** attribute provides the name of the attribute.

dt:type

The **dt:type** attribute dictates the data type of the attribute. The twenty three permitted values are:

binary	non-positive-integer
Boolean	positive-integer
byte	recurringInstant
date	short
decimal	string
double	time
float	timeDuration
Int	timeInstant
integer	unsigned-byte
long	unsigned-int
negative-integer	unsigned-long
Non-negative-integer	-

attribute

The **attribute** tag is used to associate a previously data typed attribute to a tag. This tag must be closed and please note the exact spelling with regard to case.

Type

The **type** attribute provides the data type of the custom attribute.

ElementType

The **ElementType** tag is used to declare the data type for a custom XML tag. This tag must be closed and please note the exact spelling with respect to case.

Content

The **content** attribute describes the intended content of the XML tag. There are four permitted values:

Type	Description
eltOnly	Contains only elements
empty	Contains no content
mixed	Contains both elements and text
textOnly	Contains only text

element

The **element** tag is used to associate a previously data typed tag to an element. This tag must be closed and please note the exact spelling with regard to case.

255

2.5 Creating the Animal Farm With jDOM for Java

JDOM is a proposal by a team of open source developers that distribute a free of charge DOM content management package called jDOM under the hood of jdom.org . The package is a fully functional DOM parser and DOM tree manager. Due to its simplicity and probably due to its resemblance to Microsoft's MSXML it appears to b e the current standard for XL management in Java.

Listing 2: Creating the XML farm with jDOM

```
package jdomtest;
import org.jdom.*;
import org.jdom.output.*;

public class simpledom {
  Document myDoc;
  Element root;

  public simpledom() {  } /* This is the constructor ! */

  private void makedoc() {
    root = new Element("Shop");
    myDoc = new Document(root);
  }
/* Here we add a single Animal element to our document */
  public void AddAnimal(String name, String family, int weight, String gender)  {
      Element animal = new Element("Animal");
      Attribute attName = new Attribute("Name", name);
      /* Set the name as attribute "<Animal Name="Elsa">" */
      animal.setAttribute(attName);
      root.addContent(animal);
      /* Add the family element "<Family>Cow</Family>" */
      Element elem = new Element("Family");
      elem.addContent(family);
      animal.addContent(elem);
      /* Add the Weight element and convert int to String" */
      elem = new Element("Weight");
      elem.addContent(Integer.toString(weight));
      animal.addContent(elem);
      /* Add the Gender element "<Gender>F</Gender>" */
      elem = new Element("Gender");
      elem.addContent(gender);
      animal.addContent(elem);  }

  public String XML() {
    String XMLstring;
      XMLOutputter xmlout = new XMLOutputter(" ",true);
      XMLstring = xmlout.outputString(myDoc);
      return XMLstring;
  }
/* I think toSTring is the logical name for this */
  public String toString() {
    return this.XML();
    }
  public static void main(String[] args) {
    simpledom TestSimpledom = new simpledom();
    System.out.println("Hello jDOM");
    TestSimpledom.makedoc();
    TestSimpledom.AddAnimal("Elsa", "Cow", 420, "F");
    System.out.println("*** Result from Document.toString()");
    System.out.println(TestSimpledom.myDoc.toString());
    System.out.println("*** Result from this.XML()");
    System.out.println(TestSimpledom.XML());
    System.out.println("*** Result from this.toString()");
    System.out.println(TestSimpledom.toString() );
  }
```

⇨

3 XSLT – eXtended Style Sheet Language Transformations

The eXtended Stylesheet Language is script language designed to take an XML stream as input and transform by means of predefined rules. The rules of the XSL style sheets is fully described in XML.

3.1 XSLT

260

4 SOAP and SAX

SOAP is a simple protocol to simplify remote program calls. To create a unified standard, SOAP uses XML to pass all necessary information to trigger the execute of the program on a distant server.

4.1 SOAP

4.2 SAX

265

5 UML – Unified Modelling Language

UML

UML – Unified Modelling Language

6 UML – Unified Modelling Language

270

7 Practical UML Design

8 Non-Formal Program Design Strategies

8.1 Extreme Programming and other Crash Deveolopment