# From HTTP to XML and XSLT

**U:\Book\Book_04.doc**

Database Connectivity

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Foundations of HTTP and TCP/IP Protocols</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>TCP/IP Network Protocol</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>HTTP Communication With A Webserver</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Technical Details of An HTTP Session</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Typical Protocols</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>XML – Extended Markup Language</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>XML As The Lingua Franca Of The Internet</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>The Microsoft XML Parser</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>XML – Schemas</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>XML Tags Used In Schemas</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Creating the Animal Farm With jDOM for Java</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>XSLT – eXtended Style Sheet Language Transformations</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>XSLT</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>SOAP and SAX</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>SOAP</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>SAX</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>UML – Unified Modelling Language</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>UML – Unified Modelling Language</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>Practical UML Design</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Non-Formal Program Design Strategies</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Extreme Programming and other Crash Development</td>
<td>23</td>
</tr>
</tbody>
</table>
Fehler! Formatvorlage nicht definiert/From HTTP to XML and XSLT

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 established network protocols as there has been the IBM NDIS protocol and the Novell IPX protocol.

The TCP protocol is one layer up and regroups IP packages in larger data streams. 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 address' port, the software listening to port will take the data stream and react on it to its discretion.

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>21</td>
</tr>
<tr>
<td>TELNET</td>
<td>23</td>
</tr>
<tr>
<td>HTTP</td>
<td>80</td>
</tr>
<tr>
<td>SAP R/3</td>
<td>1023</td>
</tr>
<tr>
<td>LPD or SAPLPD</td>
<td></td>
</tr>
</tbody>
</table>

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 plain ASCII protocol. Bits are grouped as tuples 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 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.

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
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>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Protocol layer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport layer</td>
</tr>
<tr>
<td>IP</td>
<td>Internet layer</td>
</tr>
<tr>
<td>Ethernet</td>
<td>Physical layer</td>
</tr>
<tr>
<td>ISDN</td>
<td></td>
</tr>
<tr>
<td>ASDL</td>
<td></td>
</tr>
<tr>
<td>HDLC</td>
<td></td>
</tr>
</tbody>
</table>

### An Example HTTP Session

GET www.logosworld.com/welcome.htm HTTP/1.0

HTTP/1.0 200 OK
Hello World

POST www.logosworld.com/shoppingcart.asp HTTP/1.0
Some data
Some data

HTTP/1.0 200 OK
Your shopping cart has been updated
1.3 Technical Details of An HTTP Session

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}
```

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
```

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
```

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
```

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/](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 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.

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

A table like this

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cow</td>
<td>420</td>
<td>F</td>
</tr>
<tr>
<td>Pig</td>
<td>120</td>
<td>M</td>
</tr>
</tbody>
</table>

could be declared as XML as follows:

Listing 1: Example of a simple XML farm

```xml
<?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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cow</td>
<td>000420F</td>
</tr>
<tr>
<td>Pig</td>
<td>000120M</td>
</tr>
</tbody>
</table>

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.
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></Animal>
<Animal><Name>Pig</Name><Weight>120</Weight><Gender>M</Gender></Animal>
```

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></Animal>
<Animal><Name>Pig</Name><Weight>120</Weight><Gender>M</Gender></Animal>
</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">
```
### 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.

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.

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.
2.3 XML – Schemas

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.

A DTD is another XML block that describes the content of the following data.
An XML Schema for the animal farm may look like this:

```xml
<?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>
```
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.

<table>
<thead>
<tr>
<th>ABAP IV</th>
<th>Visual Basic</th>
<th>XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types:</td>
<td>Public Enum TyFamily</td>
<td></td>
</tr>
<tr>
<td>TyFamily Type C(12)</td>
<td>Cow = 1 Pig = 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>End Enum</td>
<td></td>
</tr>
<tr>
<td>Types:</td>
<td>Public Enum TyGender</td>
<td></td>
</tr>
</tbody>
</table>
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

#### 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.

#### Xmins

The **xmins** 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.

#### Xmins:dt

The **xmins: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 **xmins** and the **xmins:dt** attributes exactly as displayed below:

```xml
<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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>non-positive-integer</td>
</tr>
<tr>
<td>Boolean</td>
<td>positive-integer</td>
</tr>
<tr>
<td>byte</td>
<td>recurringInstant</td>
</tr>
<tr>
<td>date</td>
<td>short</td>
</tr>
<tr>
<td>decimal</td>
<td>string</td>
</tr>
<tr>
<td>double</td>
<td>time</td>
</tr>
<tr>
<td>float</td>
<td>timeDuration</td>
</tr>
<tr>
<td>Int</td>
<td>timelInstant</td>
</tr>
<tr>
<td>integer</td>
<td>unsigned-byte</td>
</tr>
<tr>
<td>long</td>
<td>unsigned-int</td>
</tr>
<tr>
<td>negative-integer</td>
<td>unsigned-long</td>
</tr>
<tr>
<td>Non-negative-integer</td>
<td>-</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eltOnly</td>
<td>Contains only elements</td>
</tr>
<tr>
<td>empty</td>
<td>Contains no content</td>
</tr>
<tr>
<td>mixed</td>
<td>Contains both elements and text</td>
</tr>
<tr>
<td>textOnly</td>
<td>Contains only text</td>
</tr>
</tbody>
</table>
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.

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 be the current standard for XML management in Java.
Listing 2: Creating the XML farm with jDOM

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

class simplesdom {
    public simplesdom() { /* This is the constructor */ }
    private void makedoc() {
        root = new Element("Shop");
        myDoc = new Document(root);
    }
    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;
    }
    public String toString() {
        return this.XML();
    }
    public static void main(String[] args) {
        simplesdom TestSimpledom = new simplesdom();
        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
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 execution of the program on a distant server.

4.1 SOAP

4.2 SAX
5 UML – Unified Modelling Language

UML – Unified Modelling Language
6 UML – Unified Modelling Language
7 Practical UML Design
8 Non-Formal Program Design Strategies

8.1 Extreme Programming and other Crash Development