PHP and XML
Lecture-7

By: Ashiqullah Alizai
Herat University
Computer Science Faculty

Alizai.csf@hotmail.com

Binary Files

- A binary file, at its simplest, is just a stream of bits (1s and 0s).
- It's up to the application that created the binary file to understand what all of the bits mean.
- binary files can only be read and produced by certain computer programs, which have been specifically written to understand them.
  - For example, when saving a document in Microsoft Word, using a version before 2003, the file created (which has a doc extension) is in a binary format. If you open the file in a text editor such as Notepad, you won't be able to see a picture of the original Word document; the best you'll be able to see is the occasional line of text surrounded by gibberish rather than the prose, which could be in a number of formats such as bold or italic.

Text Files

- The main difference between text and binary files is that text files are human and machine readable.
- Instead of a proprietary format that needs a specific application to decipher it, the data is such that each group of bits represents a character from a known set. This means that many different applications can read text files.
  - Example: On a standard Windows machine you have a choice of Notepad, WordPad,and others, including being able to use command-line–based utilities such as Edit
  - The main disadvantage: lack of support for metadata. If you open a Word document that contains text in an array of fonts with different styles and save it as a text file, you'll just get a plain rendition; all of the metadata has been lost. What people were looking for was some way to have the best of both worlds — a human-readable file that could also be read by a wide range of applications, and could carry metadata along with its content.

A bit History

- Standard Generalized Markup Language (SGML)
  - SGML allows you to create your own markup language and then define it using a standard syntax such that any SGML-aware application can consume documents written in that language and handle them accordingly.
- HTML uses angular brackets (< and >) to separate metadata from basic text and also defines a list of what can go into these brackets, such as em for emphasizing text, tr for table, and td for representing tabular data.

Introduction to XML

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

Usage of XML

- There are two main uses for XML:
  - way to represent low-level data,
    - for example configuration files.
  - second is a way to add meta data to documents;
    - for example, you may want to stress a particular sentence in a report by putting it in italics or bold.

The Difference Between XML and HTML

- XML is not a replacement for HTML.
- XML and HTML were designed with different goals:
  - XML was designed to transport and store data, with focus on what data is
  - HTML was designed to display data, with focus on how data looks
- HTML is about displaying information, while XML is about carrying information.

XML Scenarios

- Configuration Files
- Web Services
- Web Content
- Document Management
- Database Systems
- Image Representation
- Business Interoperability

XML Technologies

- XML Parsers
- DTDs and XML Schemas : Both document type definitions (DTDs) and XML Schemas serve to describe the definition of an XML document, its structure, and what data is allowed where.
- XML Namespaces: serve as a way of grouping XM
- *XPath* is a language for accessing parts of XML documents
- *XLink* and *XPointer* support cross-references
- *XSLT* is a language for transforming XML documents into HTML like documents (including XHTML, for displaying XML files)
  - Limited styling of XML can be done with *CSS* alone
- *XQuery* is a lanaguage for querying XML documents

Main Features of XML

- No fixed set of tags
  - New tags can be added for new applications
  - An agreed upon set of tags can be used in many applications
  - *Namespaces* facilitate uniform and coherent descriptions of data
  - For example, a namespace for address books determines whether to use <te> or <phone>

Usage of XML

- XML is used in many aspects of web development, often to simplify data storage and sharing.
- XML Separates Data from HTML
  - XML simplify the process of editing data when changed in case of dynamic web development
- XML Simplifies Data Sharing
- XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.
- XML Simplifies Data Transport
- Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
-

Usage of XML

- XML Simplifies Platform Changes
  - XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
- XML Makes Your Data More Available
  - With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.
- XML is Used to Create New Internet Languages
- Here are some examples:
  - XHTML
  - WSDL for describing available web services
  - WAP and WML as markup languages for handheld devices
  - RSS languages for news feeds
  - RDF and OWL for describing resources and ontology

XML Document

- XML documents form a tree structure that starts at "the root" and branches to "the leaves".

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
<note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
</note>
```

- The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).
- The next line describes the **root element** of the document
- The next 4 lines describe 4 **child elements** of the root
- And finally the last line defines the end of the root element:

XML Syntax Rules

- The syntax rules of XML are very simple and logical.
- **All XML Elements Must Have a Closing Tag**
  - <p>This is a paragraph</p>
- **XML Tags are Case Sensitive**
  - <Message>This is incorrect</Message>
- **XML Elements Must be Properly Nested**
  - <b><i>This text is bold and italic</b></i>
- **XML Documents Must Have a Root Element**

```
<root>
    <child>
      <subchild>.....</subchild>
    </child>

</root>
```

```
<note date="12/11/2007">
    <to>Tove</to>
    <from>Jani</from>
</note>
```

- **XML Attribute Values Must be Quoted**

Entity References

- Some characters have a special meaning in XML.
- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
- This will generate an XML error:
  - <message>if salary < 1000 then</message>
- To avoid this error, replace the "<" character with an **entity reference**:
  - <message>if salary &lt; 1000 then</message>

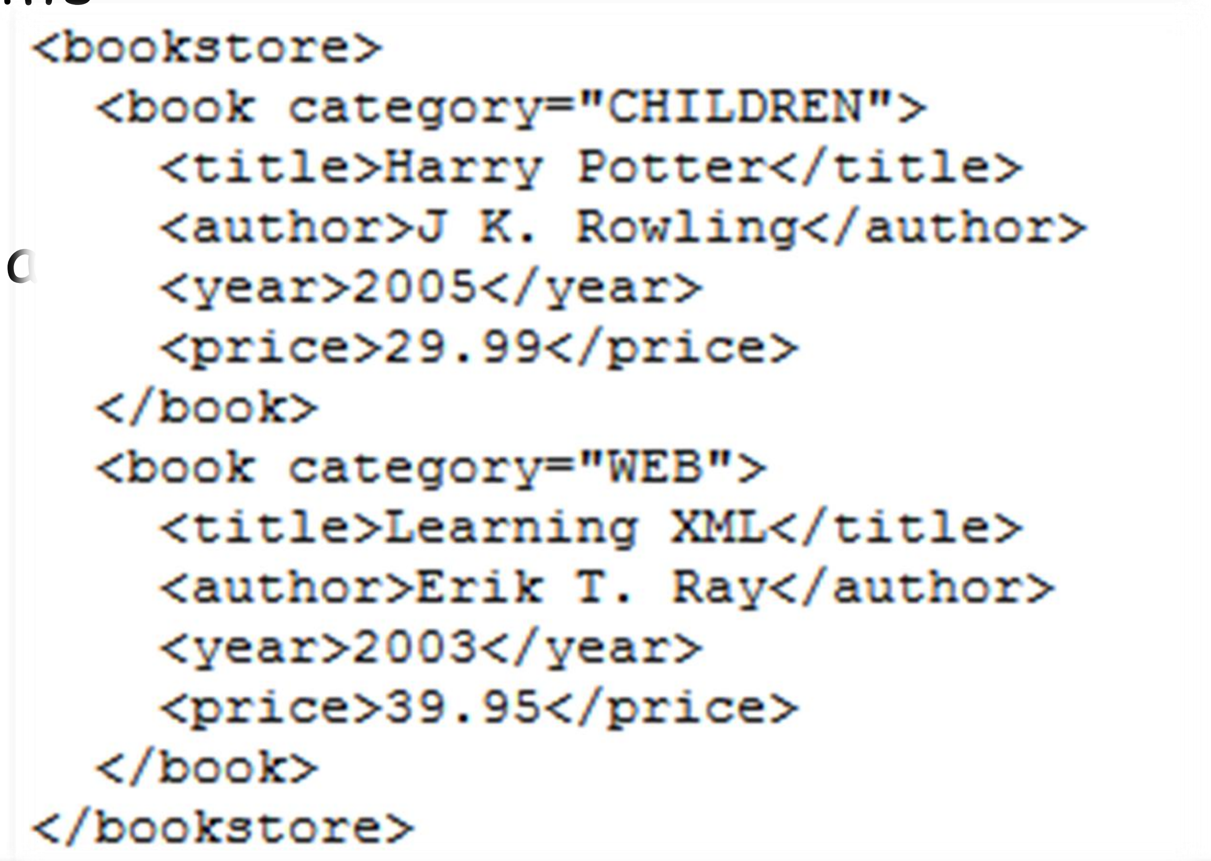
&lt;	<	less than
&gt;	>	greater than
&amp;	&	Ampersand
&apos;	'	Apostrophe
&quot;	"	Quotation mark

Comments in XML

- The syntax for writing comments in XML is similar to that of HTML.
    - `<!-- This is a comment -->`
- **White-space is Preserved in XML**
- HTML truncates multiple white-space characters to one single white-space:
    - HTML:Hello   Tove
    - Output:Hello Tove
- With XML, the white-space in a document is not truncated.
- **XML Stores New Line as LF**
- In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF).
- In Unix and Macintosh applications, a new line is stored as an LF character.

XML Elements

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain:
  - other elements
  - text
  - attributes
  - or a mix of a



```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML Naming Rules

- XML elements must follow these naming rules:
  - Names can contain letters, numbers, and other characters
  - Names cannot start with a number or punctuation character
  - Names cannot start with the letters xml (or XML, or Xml)
  - Names cannot contain spaces
  - Any name can be used, no words are reserved.
- Naming Style
  - Pascal-casing: This capitalizes separate words including the first: <MyElement />.
  - Camel-casing: Similar to Pascal except that the fi rst letter is lowercase: <myElement />.
  - Underscored names: Use an underscore to separate words: <my_element />.
  - Hyphenated names: Separate words with a hyphen: <my-element />.

XML Validation

- DTD (Document Type Definition)
- XML Schema

What is DTD

- When the vocabulary and structure of potential XML documents for a given purpose are considered together, you can talk about the type of the documents: the elements and attributes in these documents, and how they interrelate are designed to cover a particular subject of interest.
- Definitions (DTDs) are a way to describe fairly precisely the "shape" of the language.
- Like the human reading language this is also a kind of grammar to the XML file
- An XML document may have an optional DTD.
- DTD serves as grammar for the underlying XML document, and it is part of XML language.
- DTD has the form:

  <!DOCTYPE name [markupdeclaration]>

Example DTD for the XML File

```
// DTD For XML File
<?xml version=”1.0”?>
<!DOCTYPE name [
<!ELEMENT name (first, middle,
    last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT middle (#PCDATA)>
<!ELEMENT last (#PCDATA)>
]>
```

```
//XML File
<name>
<first>Ahmad</first>
<middle>Zubair</middle
<last> Ahmadi</last>
</name>
```

ANATOMY OF A DTD

- Element declarations
    - The ELEMENT declaration
    - The element name
    - The element content model
    - <!ELEMENT name (first, middle, last)>
- Attribute declarations
- The attribute name
- The attribute type
- The attribute value declaration
- <!ELEMENT contacts (contact*)>
- <!ATTLIST contacts source CDATA #IMPLIED>
- XML text is called PCDATA(parsed character data)

Element Content

- Element
- Sequences

  <!ELEMENT name (first, middle, last)>
- Choices

  <!ELEMENT location (address | GPS)>
- Mixed

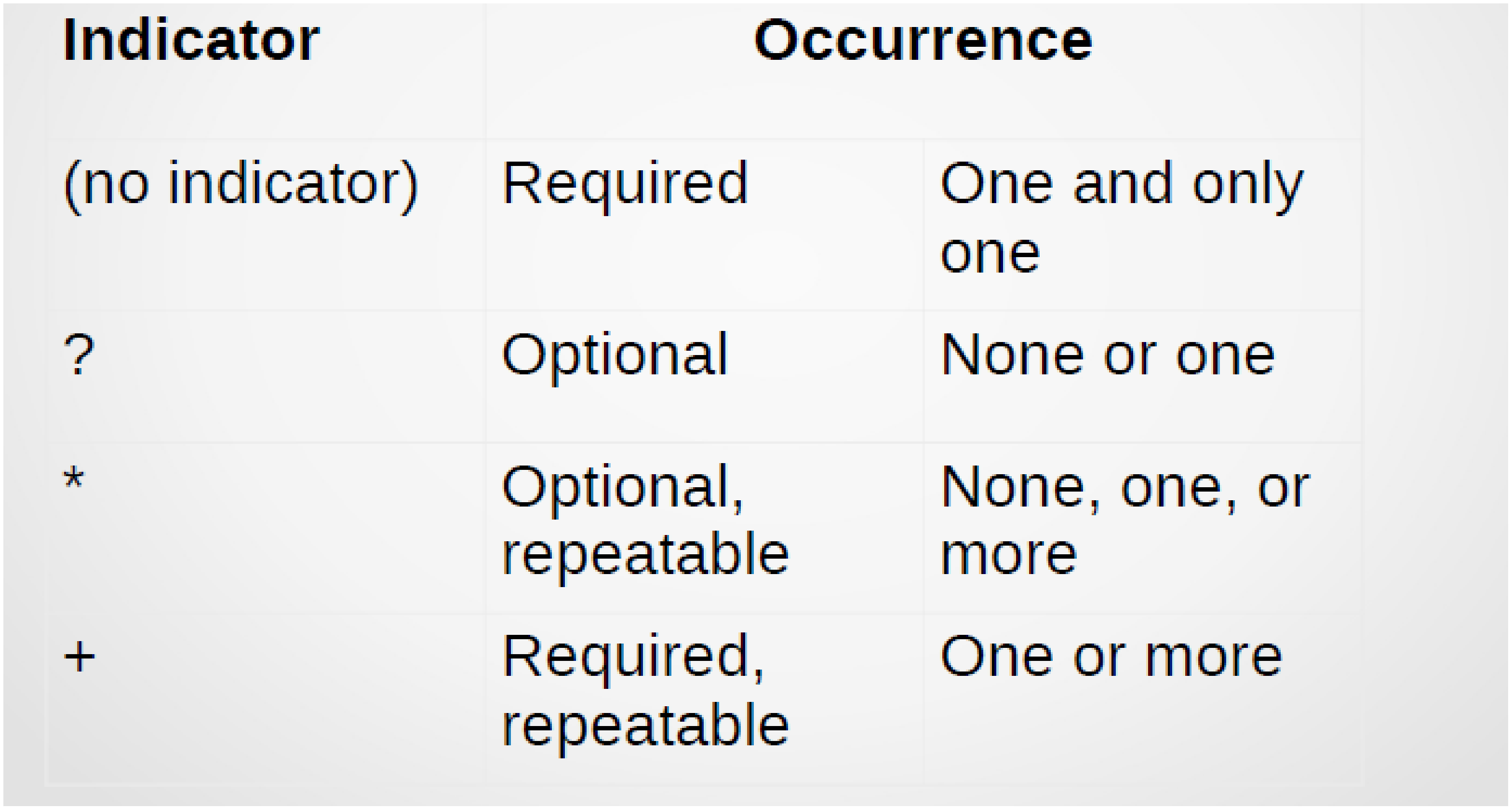
  <!ELEMENT description (#PCDATA | title | detail)*>
- Empty

  <!ELEMENT br EMPTY>
- Any

  <!ELEMENT description ANY>

DTD Indicators

Occurrence Indicator:



Indicator	Occurrence	
(no indicator)	Required	One and only one
?	Optional	None or one
*	Optional, repeatable	None, one, or more
+	Required, repeatable	One or more

DTD Example

Consider an XML document:

```
<db>
    <person>
        <name>Alan</name>
        <age>42</age>
        <email>agb@usa.net </email>
    </person>
    <person>
           .........
    </person>
        ..........
</db>
```

DTD Example

DTD for it might be:

```
<!DOCTYPE db [
    <!ELEMENT db (person*)>
    <!ELEMENT person (name, age, email)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT age (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
 ]>
```

DTD Limitation

- Poor support for XML namespaces
- Poor data typing
- Limited content model descriptions

XML Schema

- Like DTDs, XML Schemas are used for defining XML vocabularies.
- They describe the structure and content of XML documents in more detail than DTDs, and hence allow more precise validation.
- XML Schemas Use XML Syntax
    - <element name="first" type="string"/>
- Most XML Schemas are stored within a separate document in a similar fashion to external DTDs;

XML schema example

```
<?xml version=”1.0”?>

   <schema xmlns=”http://www.w3.org/2001/XMLSchema”
    xmlns:target=”http://www.example.com/name”
    targetNamespace=”http://www.example.com/name” elementFormDefault=”qualified”>

      <element name=”name”>

       <complexType>

          <sequence>

             <element name=”first” type=”string”/>

            <element name=”middle” type=”string”/>

            <element name=”last” type=”string”/>

         </sequence>

           <attribute name=”title” type=”string”/>

      </complexType>

    </element>
```

Description

- The root element within your XML Schema is the <schema> element. Within the <schema> element,you have its namespace declaration http://www.w3.org/2001/XMLSchema
- A <complexType> definition enables you to specify the allowable elements and their order as well as any attribute declarations.
- The <sequence> declaration contains three <element> declarations. Within these declarations, you have specified that their type is string. This indicates that the elements must adhere to the XML Schema simple type string, which allows any textual content.

XML Example

```
<?xml version="1.0"?>
<name
xmlns="http://www.example.com/name"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/name
   name5.xsd"
title="Mr.">
<first>Ahmad</first>
<middle> ali</middle>
<last>Alizada</last>
</name>
```

Content Models

- <sequence>: Elements must appear in the given order.
- <choice>: Only one of the elements in the list may appear.
- <all>: Elements can appear in any order, with each child element occurring zero or one time.

```
<complexType name=”NameOrEmail”>
<choice>
<element name=”email” type=”string”/>
<sequence>
<element name=”first” type=”string”/>
<element name=”middle” type=”string”/>
<element name=”last” type=”string”/>
</sequence>
 </choice>
<complexType>
```

PHP with XML

History of PHP Supporting XML

- PHP 3 - SAX (Simple API for XML) parser (Expat parser)
- PHP 4 - SAX + DOMXML
  - (non-standard, API-breaking, memory leaking, incomplete functionality)
- PHP 5 - SAX + DOM + SimpleXML

  (Re-written from scratch)

XML parser

- To read and update - create and manipulate - an XML document, you will need an XML parsers.
- There are two basic types of XML parsers:
- Tree-based parser: This parser transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements. e.g. the Document Object Model (DOM)
- Event-based parser: Views an XML document as a series of events. When a specific event occurs, it calls a function to handle it

Xpathe parser

- The Expat parser is an event-based parser.
- Event-based parsers focus on the content of the XML documents, not their structure.
- Because of this, event-based parsers can access data faster than tree-based parsers.

Event parser Structure

- Look at the following XML fraction:
- <from>Jani</from>
- An event-based parser reports the XML above as a series of three events:
- Start element: from
- Start CDATA section, value: Jani
- Close element: from
- The XML example above contains well-formed XML. However, the example is not valid XML, because there is no Document Type Definition (DTD) associated with it.

Installation

- The XML Expat parser functions are part of the PHP core.
- There is no installation needed to use these functions.

Example using Xpath parser

- **An XML File:** The XML file below will be used in our example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
```

Example

```
<?php
//Initialize the XML parser
$parser=xml_parser_create();

//Function to use at the start of an element
function start($parser,$element_name,$element_attrs)
  {
  switch($element_name)
    {
    case "NOTE":
    echo "-- Note --<br />";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
    }
  }
```

```
//Function to use at the end of an element
function stop($parser,$element_name)
  {
  echo "<br />";
  }

//Function to use when finding character data
function char($parser,$data)
  {
  echo $data;
  }

//Specify element handler
xml_set_element_handler($parser,"start","stop");

//Specify data handler
xml_set_character_data_handler($parser,"char");

//Open XML file
$fp=fopen("test.xml","r");

//Read data
while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

//Free the XML parser
xml_parser_free($parser);
?>
```

How it works:

- Initialize the XML parser with the xml_parser_create() function
- Create functions to use with the different event handlers
- Add the xml_set_element_handler() function to specify which function will be executed when the parser encounters the opening and closing tags
- Add the xml_set_character_data_handler() function to specify which function will execute when the parser encounters character data
- Parse the file "test.xml" with the xml_parse() function
- In case of an error, add xml_error_string() function to convert an XML error to a textual description
- Call the xml_parser_free() function to release the memory allocated with the xml_parser_create() function

Expat Parser Functions

- There are many:
  - (http://www.php.net/manual/en/book.xml.php)
- **xml_parser_create** — Create an XML parser
- xml_parser_get_option — Get options from an XML parser
- **xml_parser_set_option** — Set options in an XML parser
- **xml_set_character_data_handler** — Set up character data handler
- **xml_set_default_handler** — Set up default handler
- **xml_set_element_handler** — Set up start and end element handlers
- **xml_set_processing_instruction_handler** — Set up processing instruction (PI) handler

Steps towards creating

- Create an XML parser.
- Register handler functions (callback functions) with the parser. The parser will call these registered handlers when it recognizes different nodes in the XML document. Most of the application logic is implemented in these handler functions.
- Read the data from the XML file, and pass the data to the parser. This is where the actual parsing of the data occur.
- Free the parser, after the complete file has been parsed.

PHP XML DOM Parser

- The DOM parser is an tree-based parser.
- Tree-based parser: It transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements
- Good for small and medium-sized XML files
  - (loading the whole file into the memory)

PHP XML DOM Parser

- <?xml version="1.0" encoding="ISO-8859-1"?>
  - <from>Jani</from>
- **The XML DOM parses the XML above as below:**
  - Level 1: XML Document
  - Level 2: Root element tag: <from>
  - Level 3: Root element content: "Jani"

XML file

- Note.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

PHP XML DOM Code

```
<?php
     // creates a DOMDocument-Object
    $xmlDoc = new DOMDocument();
     // loads the XML from "Note.xml" into it.
    $xmlDoc->load("Note.xml");
     puts the internal XML document into a string and print it
    print $xmlDoc->saveXML();
?>
```

- The output of the code above will be:

  Tove Jani Reminder Don't forget me this weekend!

PHP XML DOM Code

```
<?php
    //Create a DOMDocument Object
    $xmlDoc = new DOMDocument();
    // use Object to load an XML Note.xml file
    $xmlDoc->load("Note.xml");
    // assign document element to a varible
    $x = $xmlDoc->documentElement;
    //Loop recursively through all elements of all childNodes and
      output the node names and contents
    foreach ($x->childNodes AS $item){
     print $item->nodeName."= ". $item->nodeValue."<br />";
      }
?>
```

PHP XML DOM Code

- The output of the code will be:

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

Note: In this example you see that there are empty text nodes between each element. When XML generates, it often contains white-spaces between the nodes. The XML DOM parser treats these as ordinary elements, and if you are not aware of them, they sometimes cause problems.

DOM Functions

- **DOM Document Functions**
- There are many:
- **http://www.php.net/manual/en/class.domdocument.php**
  - **DOMDocument::__construct** — Creates a new DOMDocument object
  - **DOMDocument::load** — Load XML from a file
  - **DOMDocument::saveXML** — Dumps the internal XML tree back into a string
- **DOM Element Functions**
  - **DOMElement::getAttribute** — Returns value of attribute
  - **DOMElement::getAttributeNode** — Returns attribute node

PHP SimpleXML

- SimpleXML handles the most common XML tasks
- It is an easy way of getting an element's attributes and text content
- Compared to DOM or the Expat parser, SimpleXML just takes a few lines of code to read text data from an element.

How it work

- SimpleXML converts the XML document into an object, such as:
- Elements - Are converted to single attributes of the SimpleXMLElement object. When there's more than one element on one level, they're placed inside an array
- Attributes - Are accessed using associative arrays, where an index corresponds to the attribute name
- Element Data - Text data from elements are converted to strings. If an element has more than one text node, they will be arranged in the order they are found

Benefits of SimleXML

- SimpleXML is fast and easy to use when performing basic tasks like:
  - Reading XML files
  - Extracting data from XML strings
  - Editing text nodes or attributes
- However, when dealing with advanced XML, like namespaces, you are better off using the Expat parser or the XML DOM.

Installation

- As of PHP 5.0, the SimpleXML functions are part of the PHP core.
- There is no installation needed to use these functions.

Using SimpleXML

- Recal Note.xml
- We need to
    - Load the XML file
    - Get the name of the first element
    - Create a loop that will trigger on each child node, using the children() function
    - Output the element name and data for each child node

PHP XML DOM Code

```
//php code SimpleXMLEx.php
<?php
  //Load the XML file
   $xml = simplexml_load_file("test.xml");
   // Get the name of the first element
   echo $xml->getName() . "<br />";
   // Create a loop that will trigger on each child node, using the children()
   function
   foreach($xml->children() as $child)
     {
  // Output the element name and data for each child node
     echo $child->getName() . ": " . $child . "<br />";
     }
   ?>
```

Output

```
note
to: Tove
from: Jani
heading: Reminder
body: Don't forget me this weekend!
```

PHP SimpleXML Functions

- simplexml_import_dom- Get a SimpleXMLElement object from a DOM node.
- simplexml_load_file- Interprets an XML file into an object
- simplexml_load_string-Interprets a string of XML into an object
- **SimpleXMLElement Class:**
- SimpleXMLElement::addAttribute -Adds an attribute to the SimpleXML element
- SimpleXMLElement::addChild - Adds a child element to the XML node
- SimpleXMLElement::asXML-Return a well-formed XML string based on SimpleXML element
- SimpleXMLElement::attributes-Identifies an element's attributes (…to be continued…)
- SimpleXMLElement::children — Finds children of given node
- SimpleXMLElement::__construct Creates a new SimpleXMLElement object
- SimpleXMLElement::count - Counts the children of an element
- SimpleXMLElement::getDocNamespaces - Returns namespaces declared in document

This is the end for this lecture