


# 3 Übung Web- und Multimedia-Engineering

Aufgabenstellung Ü3 *XML und PHP* und  
thematische Einführung

- *Überblick (Terminplan)*
- 1. Aufgabenstellung Ü3: XML und PHP
- 2. Anwendung von serverseitigen Technologien:
  - a) XML und Freunde
  - b) PHP (**P**HP: **H**ypertext **P**reprocessor)
- 3. Hilfreiches, Tipps und Links

# Terminplan / Ablauf

Woche	Datum	Übungsthema /-inhalt
KW 42	13./14.10.	keine Übung
KW 43	20./21.10.	Einführung Aufgabenstellung Ü1: HTML5 und CSS3  → Materialien für Übungsaufgabe 1: HTML5 und CSS3
KW 44	27./28.10.	Details zur Abgabe/Einreichung von Ergebnissen, Konsultation
KW 45	03.11. 9.00 Uhr	Abgabe Ü1 HTML5 und CSS3
KW 45	03./04.11.	Einführung Aufgabenstellung Ü2: JavaScript, Ajax, JSON und jQuery → Materialien für Übungsaufgabe 2
KW 46	10./11.11.	Konsultation
KW 47	17./18.11.	keine Übung
KW 48	24.11. 9.00 Uhr	Abgabe Ü2 JavaScript, Ajax, JSON und jQuery
KW 48	24./25.11.	Einführung Aufgabenstellung Ü3: XML und PHP → Materialien für Übungsaufgabe 3
KW 49	01./02.12.	Konsultation
KW 50	08./09.12.	Konsultation
KW 51	15.12. 9:00 Uhr	Abgabe Ü3 PHP und XML
KW 51	15./16.12.	Einführung Aufgabenstellung Ü4: Mobile App Development → Materialien für Übungsaufgabe 4 Version 2 → Schnittstellenbeschreibung (API)
KW 52	22./23.12.	keine Übung
KW 1	29./30.12.	keine Übung
KW 2	06./07.01.	Android Netzwerkkommunikation und Einführung inf_box_lib
KW 3	12./13.01.	Konsultation
KW 4	19.01. 9:00 Uhr	Abgabe Ü4 Mobile App Development
KW 4	19./20.01.	keine Übung
KW 5	26./27.01.	Anschluss, Feedback und Fragen
KW 6	02./03.02.	keine Übung
Woche	Datum	Übungsthema /-inhalt

- 4 Aufgabenstellungen

- Je ein Themenkomplex
- Je ca. 3 Wochen Bearbeitungszeit
- Gemeinsames Thema: inf\_box



- **Ü1:** Grundlagen client-seitige Technologien: HTML5 und CSS3
  - Grundgerüst für eine Webseite als Interface für inf\_box
- **Ü2:** Dynamische Webseiten: JavaScript, Ajax, JSON und jQuery
  - Manipulation, Animationen und dynamische Inhalte für die Webseite
- **Ü3:** Grundlagen server-seitige Technologien: XML und PHP
  - XML-Schema, XSLT, Grundlagen PHP einer inf\_box Serverkomponente
- **Ü4:** Mobile App Development: Android SDK
  - Einfache inf\_box Android-Client-App



Teil 1

## Aufgabenstellung Ü3: XML und PHP

- Basis ist die in Ü1 + Ü2 genutzte Webseite
- Aufgabe umfasst „3½“ Arbeitspakete
  1. Serverseitige Validierung von Login-Daten ★
  2. Erstellung eines XML-Schema für aktuellen infbox-Webservice ★★
  3. XSLT-Verarbeitung von Schema-fremden XML-Dokument in erweitertes infbox-XML-Schema und Ausgabe als HTML5 ★★★
  4. Adaption der Basislösung zu Admin-Interface ★

## 1. Serverseitige Validierung von Login-Daten ★

- ☐ Zugriff auf spätere Transformation soll nur mit gültigem Login möglich sein
- ☐ Eingegebene Daten gegen die in einer Textdatei vorhandenen Login-Daten validieren, dazu muss Textdatei eingelesen werden
- ☐ Passwörter liegen im System nur als Hash vor (Umwandlung notwendig)
  - Passwort = Benutzername
  - Zu verwendende Funktionen (PHP >= 5.5.0):

```
string password_hash(string $password, integer $algo [, array $options ])
```

[\[http://php.net/manual/de/function.password-hash.php\]](http://php.net/manual/de/function.password-hash.php)

```
boolean password_verify(string $password, string $hash)
```

[\[http://php.net/manual/de/function.password-verify.php\]](http://php.net/manual/de/function.password-verify.php)

## 2. Erstellung eines XML-Schema für aktuellen infbox-Webservice ★★

- ☐ Kompletter infbox-Webservice soll in einem XML-Schema beschrieben werden
  - ☐ Ergebnis / XML-Schema in *infbox\_schema\_normal.xsd* speichern (auf korrekte Benennung achten!)
- ☐ Definition von Einzeldatentypen und Listen
- ☐ Minimale und gut erweiterbare Lösung bevorzugt



## 3. XSLT-Verarbeitung von Schema-fremden XML-Dokument in erweitertes infbox-XML-Schema und Ausgabe als HTML5



- ☐ Erweiterung des Schemas aus 2 um Metadata als Element von Item (Schema muss alten und neuen WS validieren können)
  - ☐ Ergebnis / XML-Schema in *infbox\_schema\_extended.xsd* speichern (auf korrekte Benennung achten!)
- ☐ Beschreibung der Transformation von vorgegebener *file.xml* in Schema-valides Format mittels XSLT
  - ☐ Ergebnis / Transformation in *transformation.xslt* speichern (auf korrekte Benennung achten!)
- ☐ Transformation mittels des PHP-XSLT-Prozessors und Abspeichern des Ergebnis als *result.xml*
  - ☐ Auf korrektes Format von Datum und Mimetype achten!
  - ☐ Auf korrekte Benennung achten!

## 3. XSLT-Verarbeitung von Schema-fremden XML-Dokument in erweitertes infbox-XML-Schema und Ausgabe als HTML5



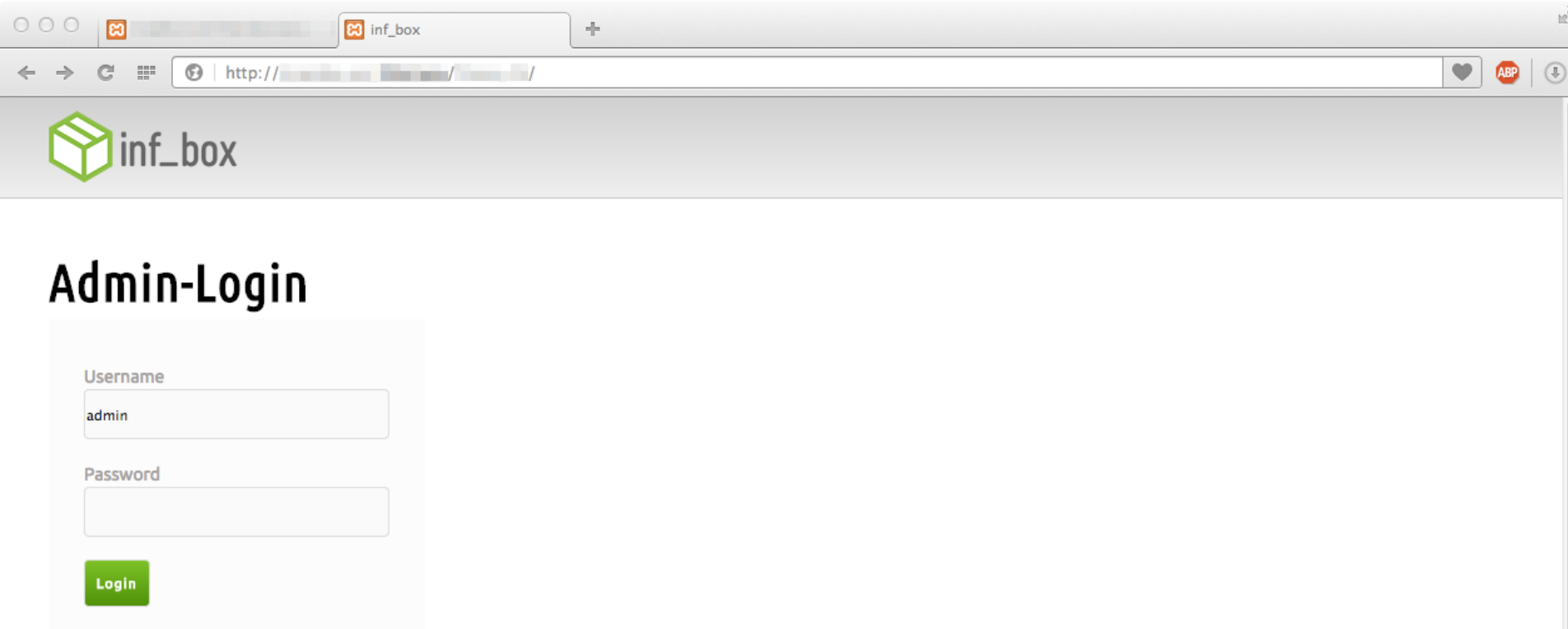
- ☐ Ausgabe des Transformationsergebnisses in Tabellenform als Bestandteil der Webseite  
(ID, Name mit Pfad verlinkt, Erstelldatum, Filetype)
- ☐ Auf HTML5-Validität achten!

## 4. Adaption der Basislösung zu Admin-Interface ★

- ☐ Entfernen von überflüssigen Menüelementen und Ersetzung durch eigene Punkte
- ☐ Tabelle durch Eingabeformular ersetzen
- ☐ **Zusatz:** Ausgabefenster direkt integrieren und AJAX-Request für die Darstellung auf der gleichen Seite nutzen

## 4. Adaption der Basislösung zu Admin-Interface

- Beispiel / Mögliche Umsetzung:



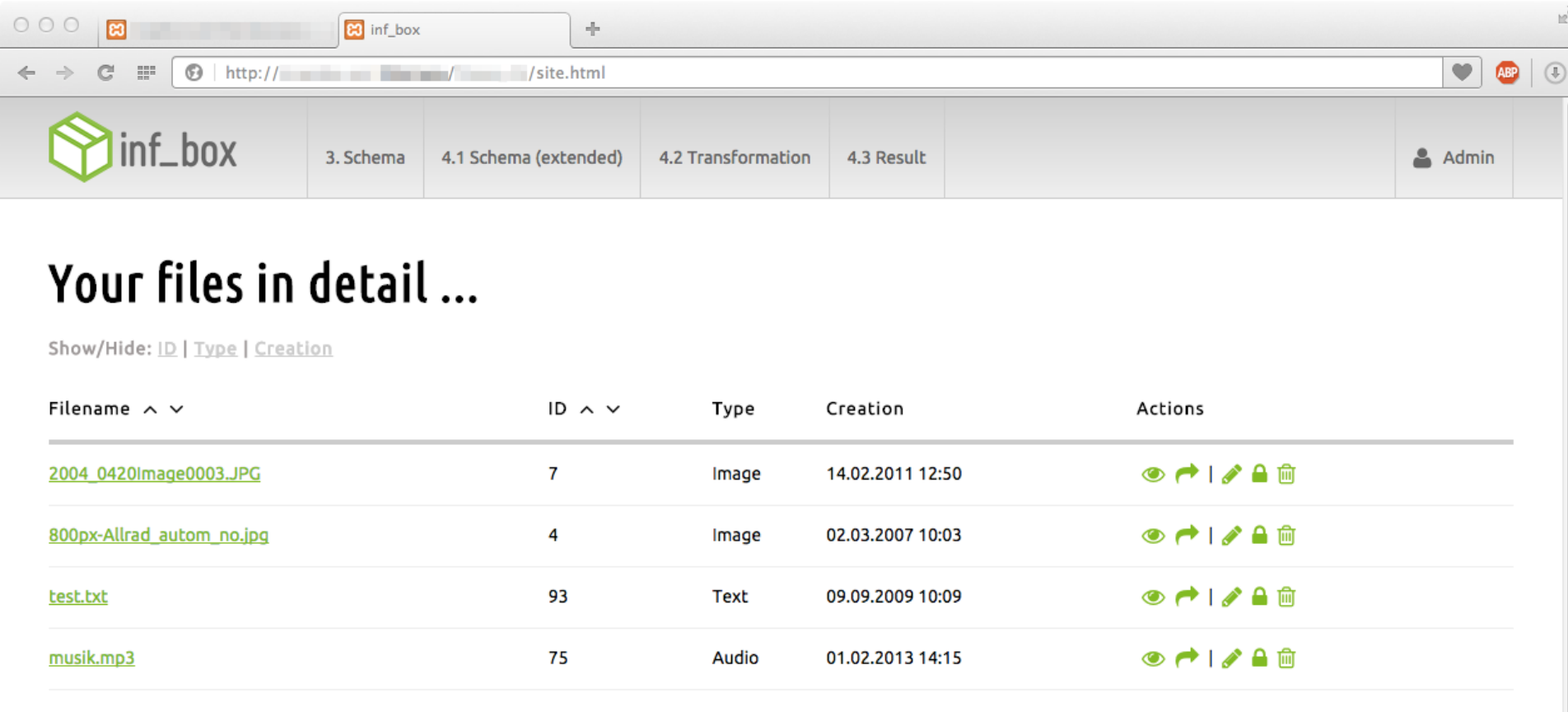
The screenshot shows a web browser window with the address bar displaying a URL. The browser's address bar shows a URL starting with 'http://'. The page header features the 'inf\_box' logo. The main content area is titled 'Admin-Login'. Below the title, there is a login form with two input fields: 'Username' (containing 'admin') and 'Password' (empty). A green 'Login' button is positioned below the password field.

### Hinweise:





















- während dem Aufrufen dieser Seite wird per PHP die Transformation von files.xml zu result.xml durchgeführt
- nach dem Login wird die HTML-Tabelle dann per AJAX aus der result.xml erzeugt (Aufgabe 1 Zusatz)

## 4. Adaption der Basislösung zu Admin-Interface

- Beispiel / Mögliche Umsetzung:



The screenshot shows a web browser window with the URL `http://.../site.html`. The browser's address bar and tabs are visible. The page header includes the **inf\_box** logo and a navigation menu with the following items: 3. Schema, 4.1 Schema (extended), 4.2 Transformation, 4.3 Result, and an **Admin** button. The main content area is titled **Your files in detail ...** and includes a link to `Show/Hide: ID | Type | Creation`. Below this is a table with the following data:

Filename ^ v	ID ^ v	Type	Creation	Actions
<a href="#">2004_0420Image0003.JPG</a>	7	Image	14.02.2011 12:50	      
<a href="#">800px-Allrad_autom_no.jpg</a>	4	Image	02.03.2007 10:03	      
<a href="#">test.txt</a>	93	Text	09.09.2009 10:09	      
<a href="#">musik.mp3</a>	75	Audio	01.02.2013 14:15	      

- Allgemeine Kriterien
  - ☐ Dateikodierung (Encoding): UTF-8 und *Unix-LF (Zeilenende)*
  - ☐ Valides HTML5, CSS3 und **XML**
  - ☐ Notwendige Header-Angaben: title, description, author, keywords
  - ☐ Fußzeile (Footer): Namen und Teamnummer
  - ☐ Dokumentation von PHP- und XML-Lösung im Code
  - ☐ Relative Adressierung verwenden (!), da Lösung in Unterordner, bspw. */root/Team\_??/*

- Erlaubte Hilfsmittel: sämtliche PHP-Funktionalität nutzbar
  - Keine weiteren PHP-Frameworks erlaubt!
- Testen: XAMPP (bspw. Version 5.6.3 mit **PHP >= 5.6.3**),  
Chrome (aktuelle Version)
- Abgabe: Montag, **15.12.2014 bis 9.00 Uhr** 

Teil 2

# **Anwendung von serverseitigen Technologien: XML und Freunde, PHP**



Teil 2a

## **XML und Freunde**

## ■ XML-Dokument

- Logischer Aufbau: Prolog + Wurzelement
- Prolog
  - Streng genommen optional, typischerweise mindestens eine XML-Deklaration
- Wurzelement
  - i.W. Elemente, Attribute und textuelle Inhalte
  - Enthält gesamte Daten des Dokuments
  - Max. ein Wurzelement, alle weiteren Inhalte darin

```
<?xml version="1.0"?>
<quiz>
  <frage>
    Wer war der fünfte
    deutsche Bundespräsident?
  </frage>
  <antwort>
    Karl Carstens
  </antwort>
  <!-- Anmerkung: Wir
    brauchen mehr Fragen -->
</quiz>
```

Quelle: [\[http://de.wikipedia.org/wiki/XML\]](http://de.wikipedia.org/wiki/XML)

**XML**

## ■ Wohlgeformtheit

- Das Dokument besteht aus mindestens einem Element
- Es gibt genau ein Wurzelement
- Alle weiteren Elemente haben ein übergeordnetes Element, in dem sie begonnen und auch enden
- Alle geöffneten Elemente werden geschlossen
- Alle Entities sind deklariert (bis auf amp, lt, gt, apos und quot)

## ■ Gültigkeit

- Ein XML-Dokument ist gültig, wenn es eine zugehörige Grammatik gibt und es dieser entspricht
- Jedes gültige XML-Dokument ist automatisch wohlgeformt

- Schemadefinition beginnt mit:
  - `<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">`  
... `</xsd:schema>`
  - Unterscheidung zwischen einfachen (simpleType) und komplexen (complexType) Datentypen
- Einfache Datentypen
  - Enthalten nur Daten, keine Unterelemente, keine Attribute
  - Sind vordefiniert, z. B. `byte`, `integer`, `short`, `float`, `date`, `time`, `duration`, `language`, ...
  - Können sowohl für Elemente, als auch für Attribute benutzt werden
  - Die Definition eigener Typen ist durch Einschränkung, Vereinigung sowie Listenbildung vordefinierter Typen möglich

# XML Schema: Einfache Typen - Beispiele

```
<xsd:simpleType name="TZylinder">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="1"/>  
    <xsd:maxInclusive value="18"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

Einschränkung von „integer“ auf Werte zwischen 1 und 18

```
<xsd:simpleType name="TDDKennzeichen">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="  
      DD-[A-Z]{2}\d{4}" />  
  </xsd:restriction>  
</xsd:simpleType>
```

Einschränkung von „string“ auf erlaubte KFZ-Kennzeichenkombinationen für Dresden durch regulären Ausdruck

# XML Schema: Einfache Typen - Beispiele

```
<xsd:simpleType name="TFahrzeuge">  
  <xsd:union memberTypes="TFahrrad TMotorrad  
    TPkw TLkw ..."/>  
</xsd:simpleType>
```

Vereinigung von existierenden  
Typen zu einem neuen Typ

```
<xsd:simpleType name="TMime-type">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="image/jpeg"/>  
    <xsd:enumeration value="image/gif"/>  
    ...  
  </xsd:restriction>  
</xsd:simpleType>
```

Einschränkung von „string“  
durch Aufzählung erlaubter  
Werte

```
<xsd:simpleType name="TMime-types">  
  <xsd:list itemType="TMime-type"/>  
</xsd:simpleType>
```

Listenbildung

- Komplexe Datentypen
  - Können Unterelemente und Attribute besitzen
  - Definition mit dem Element: `<xsd:complexType>`
  - Deklaration d. Unterelemente:  
`<xsd:element> ... </xsd:element>`
    - Als Sequenz: `<xsd:sequence> ... </xsd:sequence>`
    - Als Alternative: `<xsd:choice> ... </xsd:choice>`
    - Unterelemente in beliebiger Reihenfolge mit  
`<xsd:all> ... </xsd:all>`
    - Kardinalität mit Attributen `minOccurs` und `maxOccurs`
  - Deklaration von Attributen:
    - `<xsd:attribute ... />`
      - `use="required|optional|prohibited"`
      - `default="..."`
      - `fixed="..."`

# XML Schema: Komplexe Typen - Beispiele

```
<xsd:element name="Fahrzeug" type="TFahrzeug"/>
<xsd:complexType name="TFahrzeug">
  <xsd:sequence>
    <xsd:element name="hersteller" type="xsd:string"/>
    <xsd:element name="typ" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="raeder" type="xsd:short" default="4"/>
</xsd:complexType>
```

```
<Fahrzeug>
  <hersteller>Skoda</hersteller>
  <typ>Skoda 105L</typ>
</Fahrzeug>
```

```
<xsd:element name="Leistung" type="TLeistung" />
<xsd:complexType name="TLeistung">
  <xsd:choice minOccurs="0">
    <xsd:element name="PS" type="xsd:integer"/>
    <xsd:element name="kW" type="xsd:integer"/>
  </xsd:choice>
</xsd:complexType>
```

0 → Element ist optional

```
<Leistung>
  <kW>40</kW>
</Leistung>
```



- XPath: XML Path Language, zur XML-Pfadbeschreibung
  - Dient der Adressierung beliebiger Knoten (*nodes*) oder Knotenmengen (*node set*) innerhalb von XML-Dokumenten
  - Grundlage für XSLT
  - Operiert auf der logischen Struktur (Baum) des XML-Dokuments
  - Definition von Achsen (*axes*) + Funktionen zur Navigation
  - Knotenarten:
    - RootNode (Wurzelknoten): Nicht Wurzelement sondern dessen „virtueller“ Elternknoten
    - ElementNode (Elementknoten)
    - AttributeNode (Attributknoten), TextNode (Textknoten)
    - NamespaceNode (Namensraumknoten)
    - ProcessingInstructionNode (Verarbeitungsanweisungsknoten)
    - CommentNode (Kommentarknoten)

## ■ XPath Ausdruck

- Das primäre syntaktische Gebilde in XPath ist ein Ausdruck (Expression), der in einem bestimmten Kontext ausgewertet wird
- Auswertung eines XPath-Ausdrucks liefert eines der folgenden Objekte: Menge von Knoten, Wahrheitswert, Fließkommazahl, Zeichenkette
- Beispiel: „/buchladen/buch[preis>35.00]“

Ausdruck	Aktion im Dokument
knotenname	Selektiert alle Knoten mit Namen “knotenname”
/	Selektiert alle Knoten vom Root aus
//	Selektiert alle Knoten im Dokument, welche der Selektion entsprechen, egal wo sie liegen
.	Selektiert den aktuellen Knoten
..	Selektiert den Elternknoten des aktuellen Knotens
@	Selektiert Attribute

Quelle: [[http://www.w3schools.com/xpath/xpath\\_syntax.asp](http://www.w3schools.com/xpath/xpath_syntax.asp)]

# XSL Transformations: Grundlagen

- XSLT = Transformationssprache für XML
  - Beschreibung der Transformation eines XML-Dokumentes in eine andere Struktur
  - Filterung, Sortierung, Nummerierung und ähnliches möglich
  - Steuerung der Transformation durch **unabhängige Regeln** (keine Reihenfolge vorgegeben!)
  - Definition der Regeln über Templates
  - Template definierte die zu selektierende Elemente und die anzuwendenden Aktionen

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  <!-- created 2005-12-12-->
  <xsl:include href="xslt_stylesheet.xsl" ?>
  <xsl:output method="xml" ?>
  <xsl:template match="/" ?>
    <root>
      Heuristic: <xsl:value-of select="//text()" ?>
      <p>The leading manufact
    </root>
  </xsl:template>
</xsl:stylesheet>
```

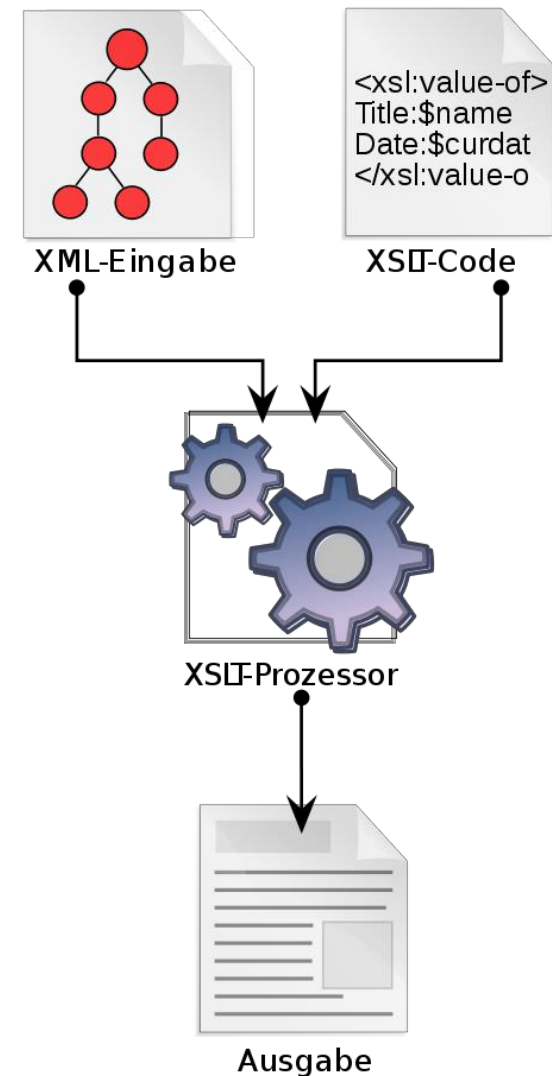
**XSLT**

Quelle: <http://de.wikipedia.org/w/index.php?title=XSLT.svg>

# XSL Transformations: Transformationsablauf

## ■ Ablauf

- XML-Dokument und XSLT-Stylesheet werden vom XSLT Prozessor geladen und verarbeitet
- Prozessor sucht nach passenden Transformationsregeln (Templates) im Stylesheet und wendet diese auf XML an
- Matching über Tag:  
`<xsl:template match="...">`  
...  
`</xsl:template>`



Quelle: <http://de.wikipedia.org/w/index.php?title=Datei:TempDeXslt015.svg>

# XSL Transformations: Beispiel

## ■ Beispiel (XML + XSLT Stylesheet)

```
<catalog>
  <cd>
    <title>Empire
      Burlesque</title>
    <artist>Bob Dylan</artist>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
  </cd>
</catalog>
```

Link zum Beispiel:

[\[http://www.w3schools.com/xsl/tryxslt.Asp?xmlfile=catalog&xsltfile=catalog\]](http://www.w3schools.com/xsl/tryxslt.Asp?xmlfile=catalog&xsltfile=catalog)

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" ... >
  <xsl:template match="/">
    <html> <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr>
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of
              select="title"/></td>
            <td><xsl:value-of
              select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table> </body> </html>
    </xsl:template>
  </xsl:stylesheet>
```

# XSL Transformations: Wichtige Tags

## ■ Wichtige Tags

- `<value-of>` Liest den Inhalt eines Knotens aus und fügt ihn ins Ausgabedokument ein
- `<for-each>` Selektion und traversieren eines *node sets*
- `<sort>` Sortieren der selektierten Knotenmenge
- `<if>` Einfache konditionale Prüfung
- `<choose>` In Verbindung mit `<xsl:when>` und `<xsl:otherwise>` für multiple konditionale Prüfungen

Teil 2b

# **Einführung PHP**

(serverseitige Programmierung)

# Einführung PHP: Blick über den Tellerrand

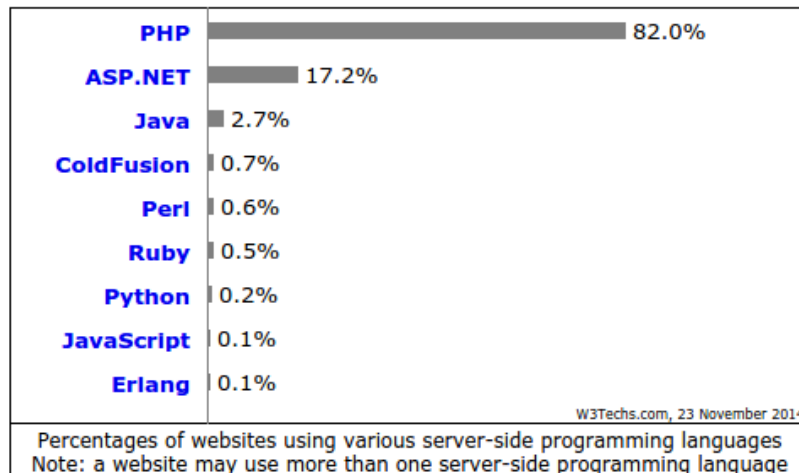
## ■ Andere Skriptsprachen

- Perl, 1987
- Python, 1991
- CGI, 1993
- PHP, 1995
- Ruby, 1995
- ASP, 1996
- JSP, 1999



TIOBE Index November 2014

Pos	Language	Rating
1	C	17.47%
2	Java	14.39%
3	Objective-C	9.063%
4	C++	6.098%
5	C#	4.985%
6	PHP	3.043%
7	Python	2.589%
8	JavaScript	2.088%
9	Perl	2.073%
10	Visual Basic .NET	2.061%



Usage of server-side programming languages for websites, W3Techs, November 2014



- Gutes Nachschlagewerk: SELFPHP, [\[http://www.selfphp.de/\]](http://www.selfphp.de/)
- „**P**HP: **H**ypertext **P**reprocessor“
  - Serverseitig interpretierte Skriptsprache, Open source
  - Syntax angelehnt an C und Perl
- Einbettung in HTML-Code und serverseitige Ausführung
  - Skripte werden vor Auslieferung interpretiert, übersetzt
- Seit PHP 4 mit Objektkonzept, Objektorientierung
- Breite Unterstützung verschiedener Datenbanksysteme
- Gängige Dateiendungen: **.php** | .php3 | .php4 | .phtml

# Einführung PHP: Funktionsweise

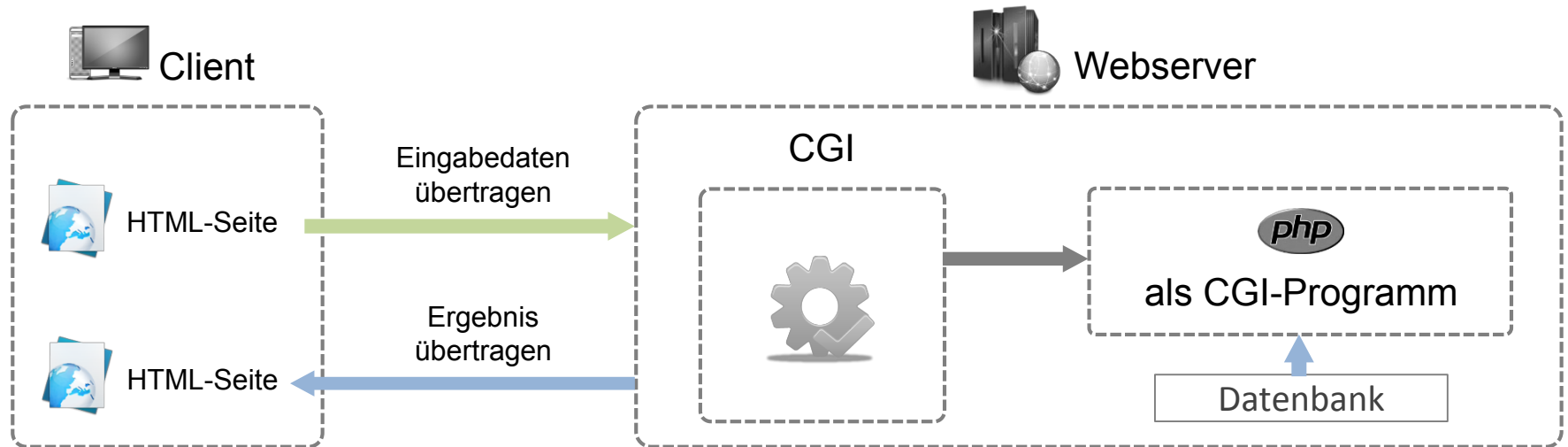
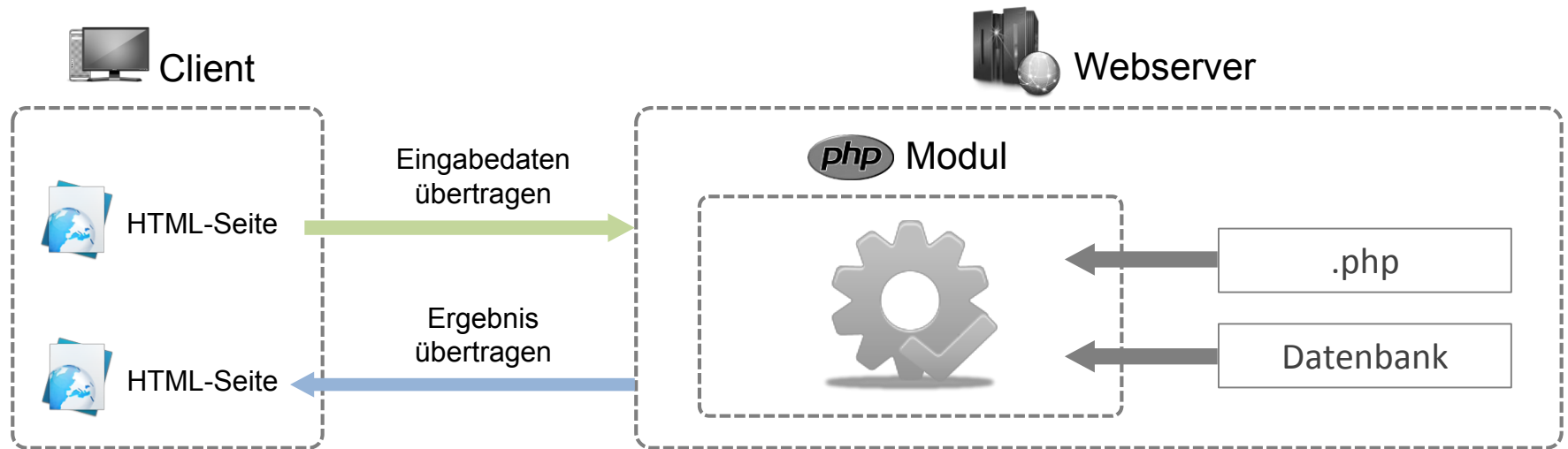


Abbildung nach: <http://selfphp.de/praxisbuch/praxisbuch.php?group=4>

- PHP-Interpreter durchsucht HTML-Code nach Anfangs- und Abschluss-Verarbeitungsinstruktionen `<?php` und `?>`
- PHP-Code Einbettung in HTML-Code
  - XML-Stil (wird am meisten genutzt, ist „sauber“)

```
<?php echo "Einbindung in XML-Stil"; ?>  
<?PHP echo "Einbindung in XML-Stil"; ?>
```

```
<?php  
    echo "Einbindung in XML-Stil";  
?>
```

- Short-Tag, funktioniert nicht immer!

```
<? echo "Einbindung in XML-Stil"; ?>
```

- Javascript-Stil

```
<script language="php">  
    echo "Einbindung im JavaScript-Stil";  
</script>
```

## ■ Minimalbeispiel:

PHP-HTML-Code auf dem Server

```
<html>
  <head>
    <title>KP MI</title>
  </head>
  <body>
    <?php echo "Hello World!"; ?>
  </body>
</html>
```



HTML-Code im Browser (Client)

```
<html>
  <head>
    <title>KP MI</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

## ■ Allgemeine Syntax

- Ende eines Befehls (o. Befehlszeile) wird mit Semikolon markiert
- Variablennamen beginnen mit \$

```
<?php
    $var = "Hello World";
    echo $var;
?>
```

## – Kommentare

```
<?php
    // Alles hinter dieser Markierung ist ein Kommentar
    # Alles hinter dieser Markierung ist ein Kommentar

    /* Alles zwischen dieser Markierung ist ein Kommentar */
    /*
        Alles zwischen dieser Markierung ist ein Kommentar
    */
?>
```

## ■ Allgemeine Syntax

- Primitive Datentypen: PHP entscheidet zur Laufzeit über Datentyp

```
<?php
    // boolean
    $isPublic = TRUE; // TRUE or FALSE
    $isPublic = 1; // 1 or 0

    // integer, float
    $x = 10;
    $y = 10.123456;

    // string
    $text = "Hello World";

    // array, object
    $num_arr = array( 10, 11, 234, 23 );
    $key_arr = array( "key_1" => 10, "key_2" => TRUE, "key_3" => „Hi" );

    $obj = new ClassName();
?>
```

## ■ Funktionsaufrufe

- Am Beispiel „Einbindung externer Skripte“

```
<?php
    include( "datei.inc" );
?>
```

- Einbindung ermöglicht Verteilung von Code, Programmstruktur
  - Stichwort: zentrale und oft verwendete Funktionen/Klassen
- include() und require() ähnliche Funktion
  - include() wirft „Warning“ bei fehlender Datei
  - require() bricht „Fatal Error“ ab
- Seit PHP 4, include\_once() und require\_once()
- Benennung der Dateien: Sicherheitsproblem!
  - Auslieferung von Dateien mit bestimmten Dateiendungen

## ■ Definition von Funktionen

- Schlüsselwort „function“

```
<?php
// Definition
function quadratSumme( $value ) {
    $result = $value * $value;
    return $result;
}

// Aufruf: Ergebnis ist 25
echo "Funktion quadratSumme liefert: " . quadratSumme(5);
?>
```

- return-Wert optional, Angabe Datentyp nicht notwendig
- Übergabe von Parametern: „by value“ (Standard) oder „by reference“ (`<?php &$var ?>`)



## ■ Bedingungen

### – if-elseif-Anweisung

```
<?php
    if ($value >= 100) { ... }
    elseif ($value >= 50) { ... }
    else { ... }
?>
```

### – Switch-case

```
<?php
    switch ($value) {
        case 100:
            ...
            break;
        case 150:
            ...
            break;
        default:
            ...
            break;
    }
?>
```

## ■ Schleifen

### – while-Schleife

```
<?php
    while () { ... }
?>
```

### – do-while-Schleife

```
<?php
    do { ... } while ();
?>
```

### – for-Schleife

```
<?php
    for ($i=0;$i<10;$i++) { ... }
?>
```

### – foreach-Schleife

```
<?php
    foreach ($array as $value) { ... }
    foreach ($array as $key => $value) { ... }
?>
```

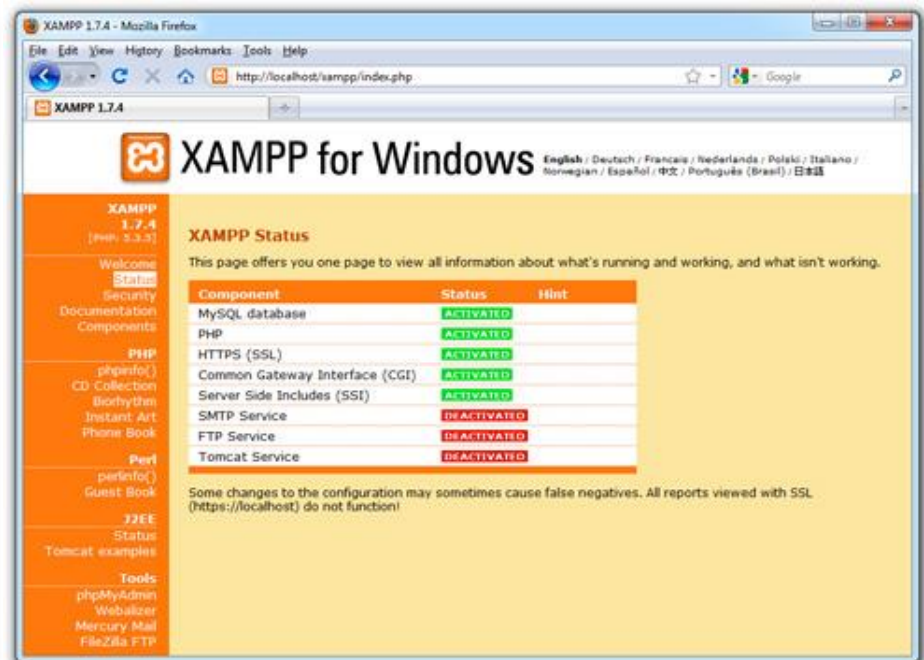
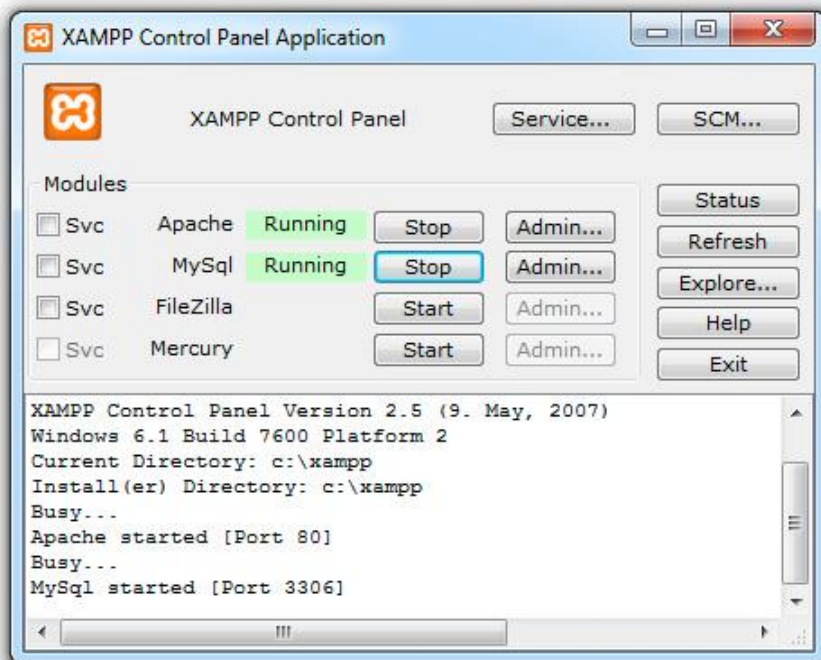
- Auszug PHP-Funktionsumfangs:
  - Array-Funktionen (Sortieren, Suche, ...)
  - Dateisystem-Funktionen (Lesen, Schreiben, Listen, ...)
  - Datums- und Zeit-Funktionen
  - Mail-Funktion
  - Mathematische-Funktionen (Winkel, Runden, ...)
  - MySQL-Funktionen (Verbindung, Schreiben, ...)
  - PDF-Funktionen (Öffnen, Einfügen, Zeichnen, Speichern, ...)
  - Session-Funktionen (Starten, Daten Speichern, ...)
  - String-Funktionen (Suche, Teilen, ...)
  
- Gutes Nachschlagewerk: SELFPHP, [\[http://www.selfphp.de/\]](http://www.selfphp.de/)

Teil 3

## **Hilfreiches, Tipps und Links**

# Hilfreiches, Tipps und Links

- XAMPP laden und entpacken
  - Website: <https://www.apachefriends.org/de/>
  - Aktuelle Version für alle Systeme (Windows, MacOS X, Linux)
  - XAMPP: **C**ross **A**pache HTTP Server, **M**ySQL, **P**HP and **P**erl
  - Inklusive phpMyAdmin (DB-Verwaltung), FileZilla FTP Server, ...



- XAMPP: Wo lege ich meine Dokumente hin?
  - Ordner im XAMPP-Verzeichnis für alle Web-Dokumente lautet `\htdocs\Team_??\`
  - Dort z.B. eine Datei *test.html* anlegen → diese dann mit der URI *http://localhost/Team\_??/test.html* aufrufen
- FAQ XAMPP für Windows
  - [\[https://www.apachefriends.org/faq\\_windows.html\]](https://www.apachefriends.org/faq_windows.html)

# Fragen?



**Interactive Media Lab Dresden**  
Professur für Multimedia-Technologie

*Kontakt:*

Ricardo Langner ([ricardo.langner@tu-dresden.de](mailto:ricardo.langner@tu-dresden.de))

Tom Horak ([tom.horak@tu-dresden.de](mailto:tom.horak@tu-dresden.de))

# Changelog

---

**Datum / Zeit**

**Beschreibung**

2014-11-24 9:15

- Initiale Downloadversion
-