Inhalt

1. Einführung in die Informatik

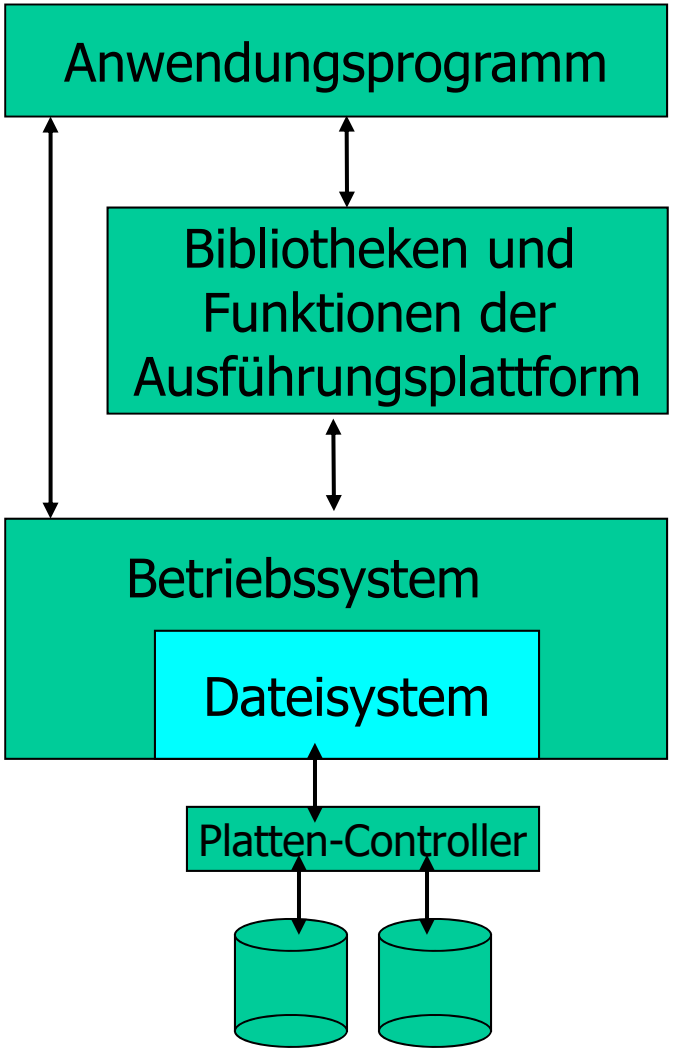
2. Algorithmen

3. Imperative Programmierung

4. Verarbeitung externer Datenquellen

- Dateien
- Office-Objekte
- Web und XML
- Exkurs: Web-Nutzerschnittstellen (ASP.NET)

Dateien - Speichersystem (1)



Anwendung liest, verarbeitet und speichert Daten

Pufferung, Zusatzfunktionen wie z.B. Dialoge zum Auswählen einer Datei

Betriebssystem stellt einheitliche Schnittstelle zu Dateisystemen bereit

Dateisystem realisiert lineare Blockadressierung, Verzeichnisse, Zugriffsrechte u. mehr

Daten werden durch Dateisystem auf ein blockorientiertes Gerät abgebildet

Gerätetreiber und Plattencontroller sprechen Platten und dort jeweils Blöcke (Zylinder, Sektor, Kopf) an.

Dateien - Speichersystem (2)

Eine Datei ist ein benannter Bereich (mit Dateinamen und Zusatzinformationen) in den Daten geschrieben werden können.

- Daten werden innerhalb einer Datei auf nummerierten Blockadressen abgelegt, beim aufeinanderfolgenden Schreiben wird die Blockadresse automatisch erhöht (Datei-Postionszeiger)
- Daten können innerhalb einer Datei adressiert werden (Setzen des Datei-Positionszeigers)
- Daten können nacheinander aus einer Datei gelesen werden, Datei-Positionszeiger wird automatisch weitergesetzt.
- Eine Datei kann vergrößert werden, wenn zusätzliche Daten geschrieben werden.
- Dateiinhalte können überschrieben werden.
- Eine Datei kann verkleinert werden.
- Vor dem Schreiben und Lesen muss eine Datei geöffnet werden.
- Sind alle Dateioperationen abgeschlossen, wird die Datei geschlossen

Dateizugriff aus Visual Basic Programmen (1)

Vor den Zugriff muss eine Datei geöffnet werden. Es können mehrere Dateien gleichzeitig geöffnet sein.

Öffnen einer Datei (Open):

FileOpen(1,“meinedaten.txt“, OpenMode.Input)

Nach Open kann auf eine Datei über die angebene Nummer (hier 1) zugegriffen werden.

Allgemeine Form

FileOpen(filenr As Integer, filename As String, Mode As Microsoft.VisualBasic.OpenMode)

Schliessen der Datei (Close)

FileClose(1)

Allgemeine Form:

FileClose(filenr As Integer)

Dateizugriff aus Visual Basic Programmen (2)

Zugriffsfunktionen für Textdateien

Input-Funktionen:

ziel = LineInput (filenr)

Liest eine Zeile aus der mit *filenr* angegebenen Datei bis zum nächst folgenden Zeilenumbruch. Die gelesenen Zeichen werden als String zurückgegeben.

Allgemeine Form:

LineInput(filenr As Integer) As String

zeile = InputString(filenr , 20)

Liest eine festgelegte Anzahl Zeichen aus der Datei (hier 20)

Allgemeine Form:

InputString(filenr As Integer, count As Integer) As String

Dateizugriff aus Visual Basic Programmen (3)

Mittels *EOF(filenr)* wird getestet, ob das Dateiende schon erreicht ist.

Allgemeine Form:

EOF(filenr As Integer) As Boolean

Schreiben in eine geöffnete Textdatei mittels *PrintLine*

Dim textzeile As String = "Der grüne Frosch springt vom Blatt."
PrintLine(13, textzeile)

Allgemeine Form:

PrintLine(filenr As Integer, ParamArray Output() As Object)

Beispiel

Der Code-Ausschnitt zeigt das zeilenweise Kopieren von Textdateien

```
Dim zeile As String
FileOpen(12,“eingabe.txt“,OpenMode.Input)
FileOpen(13,“ausgabe.txt“,OpenMode.Output)
While (Not EOF(12))
        zeile=LineInput(12)
        PrintLine(13,zeile)
End While
FileClose(12)
FileClose(13)
```

Formatierte Ausgabe auf Textdateien

Das Unterprogramm Print erlaubt eine formatierte Ausgabe mehrerer Werte in eine Datei.

Beispiel: Schreiben einer Textdatei in Visual Studio 2010:

```
FileOpen(1, "C:\TEMP\vbadatei.txt", OpenMode.Output)
Print(1, "Peter Sobe", 99, vbCrLf)
Print(1, "Max Meier", 103)
FileClose(1)
```

Allgemeine Form:

```
Print(filenr As Integer, ParamArray Output() As Object)
```

Binärdateien

Binärer Zugriff: Daten werden in ihrem internen Darstellungsformat in Datei geschrieben, bzw. aus ihnen gelesen.

Wie üblich werden Dateien vor allen Zugriffen mit Open geöffnet und am Schluss mit Close geschlossen.

FileOpen(3, "C:\TEMP\bindatei.bin", OpenMode.Binary, OpenMode.Output)

Operationen:

FileGet(filenr, variable, [*Recordnr oder Bytepostion])*

Lesen von einer Datei auf eine Variable. As Variable können alle Typen, außer Objekte und Felder angegeben werden.

Wird Recordnr, Byteposition nicht angegeben, wird vom aktuellen Dateizeiger beginnend gelesen

FilePut(filenr, variable, [*Recordnr oder Bytepostion])*

Schreiben auf Datei. Parameter wie Get.

Lesen einer Textdatei mit objektorient. Technik

Klasse StreamReader mit Methode ReadLine

```
Imports System.IO
Dim fs As FileStream
Dim sr As StreamReader
Dim dateiname As String = “meineDatei.txt"
Dim zeile As String

If Not File.Exists(dateiname) Then
   MessageBox("Dateiexistiert nicht")
Else
   fs = New FileStream(dateiname, FileMode.Open)
   sr = New StreamReader(fs)
   Do Until sr.Peek() = –1
      zeile = sr.ReadLine()
      Textstring &= zeile & vbCrLf
   Loop
sr.Close()
```

Schreiben einer Textdatei mit objektorient. Technik

Klasse StreamWriter mit Methode WriteLine

```
Imports System.IO
Dim fs As FileStream
Dim sw As StreamWriter
Dim dateiname As String = "C:\Tmp\aus.txt"

Try
   fs = New FileStream(  dateiname, FileMode.Create)
   sw = New StreamWriter(fs)
   sw.WriteLine(EingabeText)  'vorausgesetzt EingabeText wurde vorab
                              'vom Programm mit Inhalt versehen
   sw.Close()
Catch ex As Exception
   MessageBox(ex.Message)
End Try
```

Beispiele angelehnt an: Einstieg in Visual Basic 2010 von Thomas Theis, Galileo Computing

MS-Office-Objekte

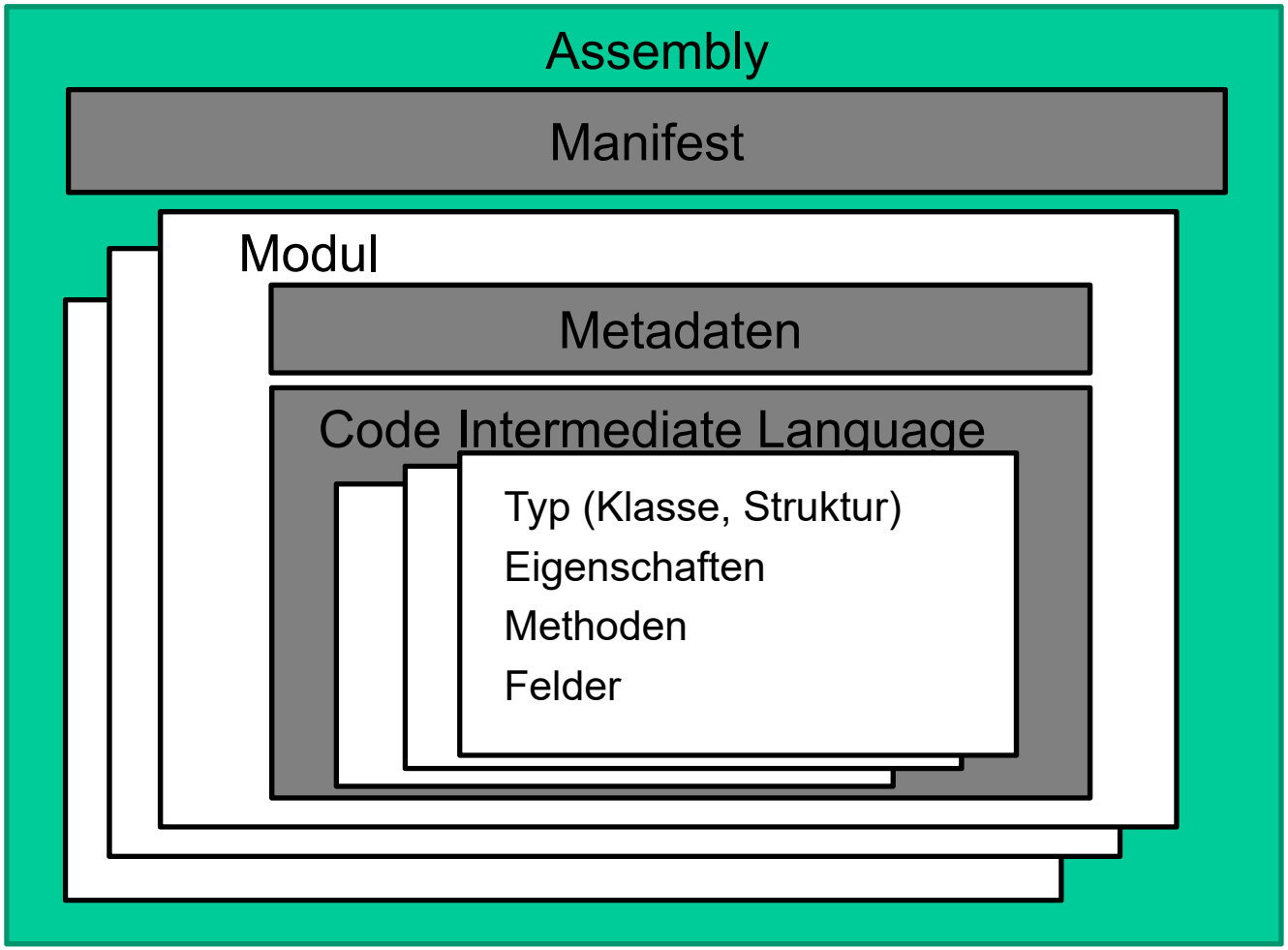
Durch .NET Programme (z.B. Visual Basic) können Microsoft-Office-Anwendungen automatisiert werden.

Diese Technik basiert auf den s.g. Interop-Assemblys – das sind Sammlungen von Funktionen und Objekten, die gemeinsam durch verschiedene auf der .NET Plattform ausgeführten Anwendungen benutzt werden können.

Installierte Assemblys sind gelistet unter C:\Windows\assembly

Interoperabilität mit Office-Anwendungen (1)

Assemblys, verwaltet von der CLR



Interoperabilität mit Office-Anwendungen (2)

Ein Beispiel für MS-Excel Automatisierung findet man unter
http://microsoft.com/kb/301982

In Visual Studio unter Projekt:

- Verweis hinzufügen
- .NET Plattform auswählen
- Assembly *Microsoft.Office.Interop.Excel* auswählen

Zugriff auf ein Excel-Dokument (1)

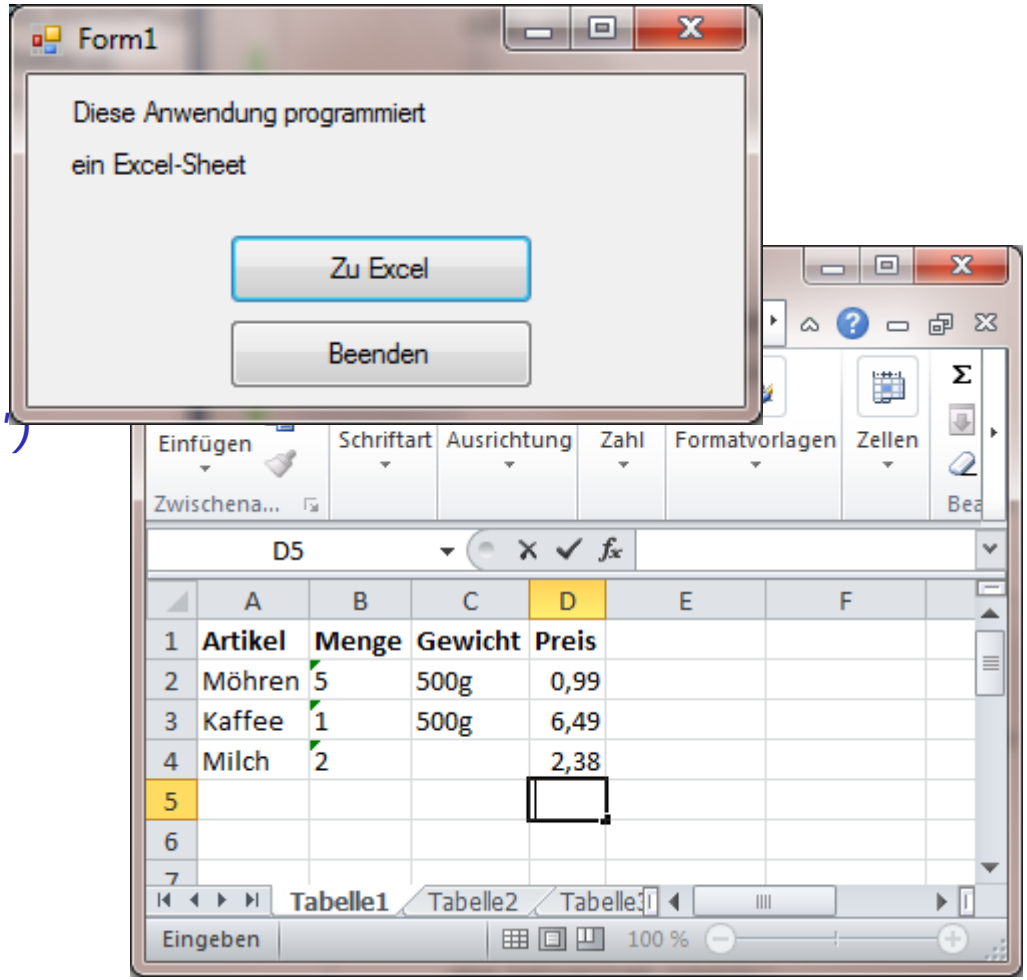
```
Imports Microsoft.Office.Interop

Dim  oXL As Excel.Application
Dim  oWB As Excel.Workbook
Dim  oSheet As Excel.Worksheet

oXL = CreateObject("Excel.Application")

oWB = oXL.Workbooks.Add
oSheet = oWB.ActiveSheet

oSheet.Cells(1, 1).Value = "Artikel"
oSheet.Cells(1, 2).Value = "Menge"
oSheet.Cells(1, 3).Value = "Gewicht"
oSheet.Cells(1, 4).Value = "Preis“
```



Nach Art dieses Beispiels können z.B. Berechnungsdaten als Tabelle ausgegeben werden und danach Diagramme erstellt werden.

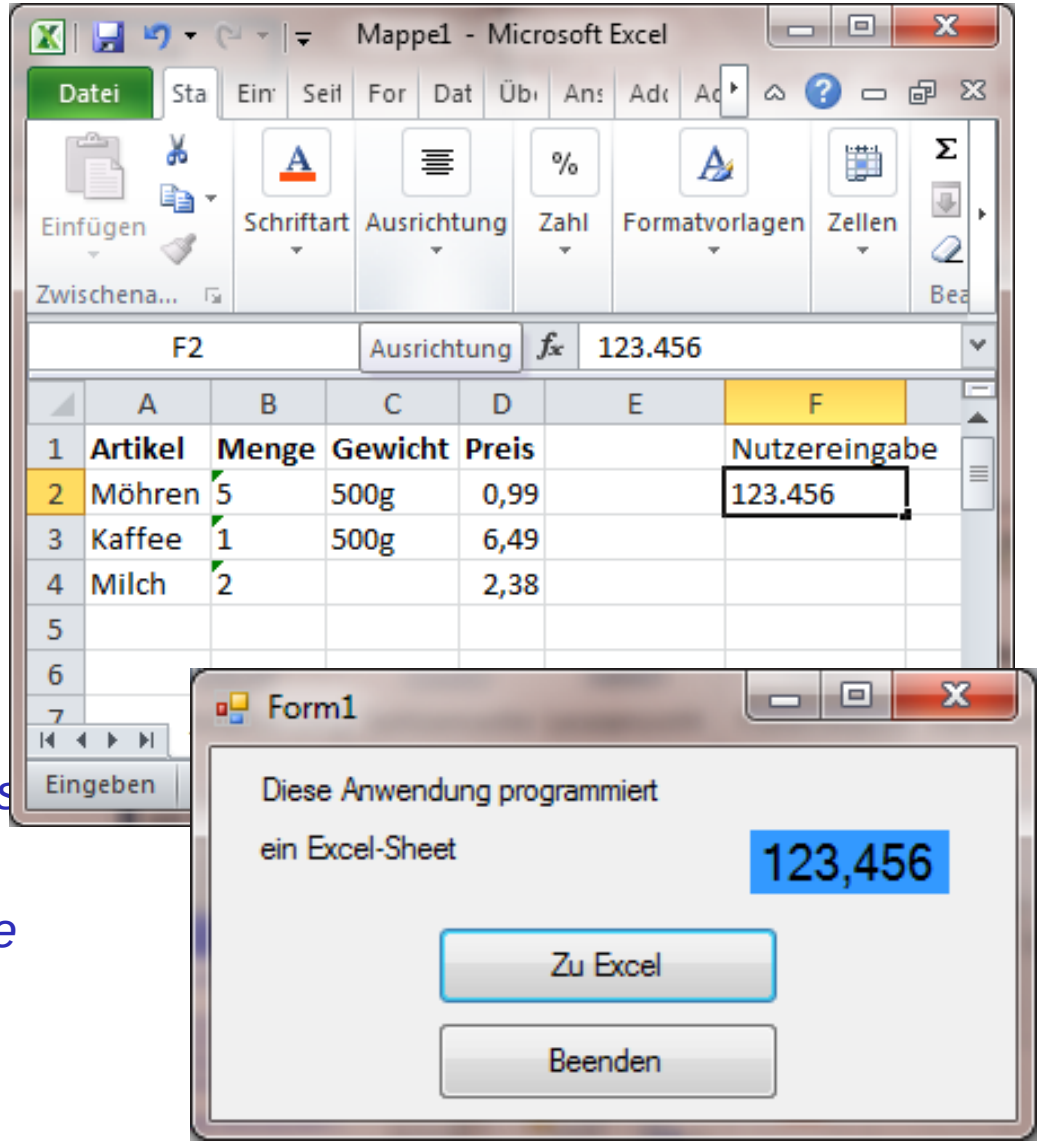
Zugriff auf ein Excel-Dokument (2)

```
Imports Microsoft.Office.Interop

Dim  oXL As Excel.Application
Dim  oWB As Excel.Workbook
Dim  oSheet As Excel.Worksheet
Dim  oRng As Excel.Range
Dim  eingabe As String

MsgBox("Geben Sie jetzt in F2 etwas

eingabe = oSheet.Range("F2").Value
Display.Text = eingabe
```



Zugriff auf Excel-Tabellen

Bei den bisherigen Beispielen wurde eine neue Excel-Tabelle erzeugt. Der Nutzer kann diese unter einem neuen Dateinamen speichern.

Auch das Öffnen bereits existierender Dokumente ist möglich:

```
Const NVAL As Integer = 20
Dim eingabe(NVAL) As Single
Dim  oXL As Excel.Application
Dim  oWB As Excel.Workbook
Dim  oSheet As Excel.Worksheet
oXL = CreateObject("Excel.Application")
oWB = oXL.Workbooks.Open(“C:\usr\tabellen\experimente.xls“)
oSheet = oWB.ActiveSheet
 REM Einlesen der Spalte (B1:B20)
FOR i=1 TO NVAL
  eingabe(i) = oSheet.Range(„B"+Format(i)).Value
NEXT i
```

Web und XML

- HTTP – Protokoll
- HTML – Format
- XML -Sprache(n) und Werkzeuge
- XML-Verarbeitung mit Visual Basic

HTTP

HTTP: Hypertext Transport Protocol

Roy Fielding, Tim Berners-Lee ab 1989

Protokoll zum Transport von Web-Inhalten, insbes. HTML-Seiten

HTTP	Bestimmt, was angefragt wird und welche Daten zurückgesendet werden
TCP	Erzeugt Verbindung zwischen Client und Server
IP	Ermöglicht Adressierung, Paket-Transfer

HTTP-Nachrichten bestehen aus Request- und Response-Nachrichten

Request-Typen (Operationen):

OPTIONS, **GET**, HEAD, **POST**, PUT, DELETE, TRACE, CONNECT

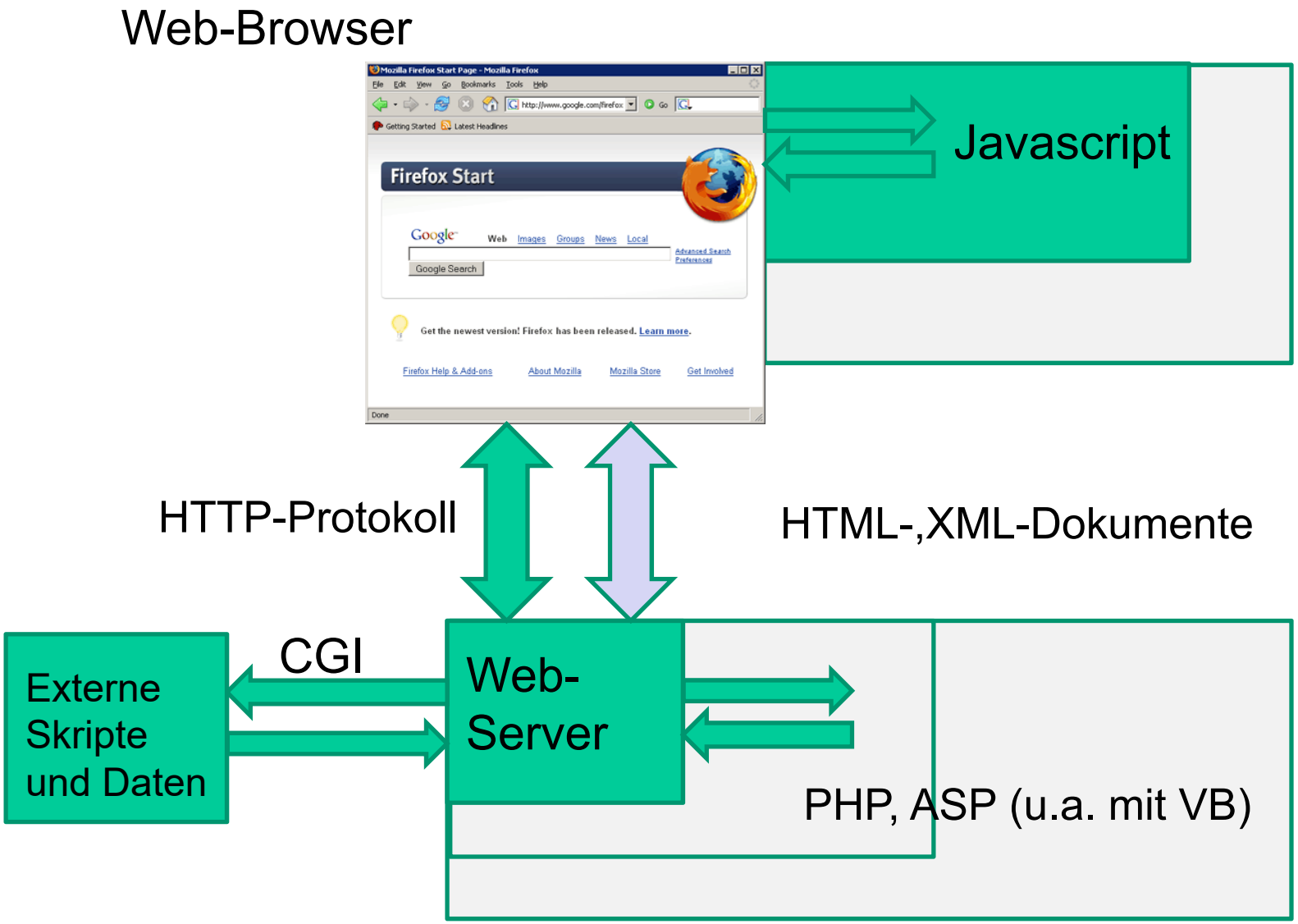
HTTP

GET- Methode - Die GET-Methode ist die am häufigsten eingesetzte Methode bei einer Client-Anforderung. Damit wird ein Dokument beim Webserver zum Herunterladen angefordert. Der Name des Dokumentes und eventuell sein Pfad müssen in der Request-URI angegeben sein.

Beispiel: ***GET /beispiel.html HTTP/1.0***

POST-Methode - Übertragung von Datenpaketen vom Client zum Server. Hauptsächlich wird sie eingesetzt, wenn ein angefordertes HTML-Dokument Formularelemente für Nutzereingaben enthält. Die eingegebenen Daten werden dann an ein serverseitig vorhandenes Programm (Script) zur Weiterverarbeitung gesendet (serverseitiges Scripting).

Web-Anwendungen unter Nutzung von HTTP



HTML

- Kodierungssprache für Webseiten.
- HTML-Dokument wird im HTTP-Message-Body

- Beispiel:

```
<HTML>
<HEAD><TITLE>DOM</TITLE></HEAD>
<BODY BGCOLOR="yellow">
 <H2>Demonstration  DOM</H2>
 <P>
  <IMG SRC="Comp2.gif" />  Bild 4.1
 </P>
 <P><I>Ende</I>
 </P>
</BODY>
</HTML>
```

HTML

Elemente: *<TITLE>DOM</TITLE>*

Attribute: *<IMG SRC="Comp2.gif" />*

Kombination Element mit Attributen:

<BODY BGCOLOR="yellow"> … </BODY>

Die Elemente

- werden durch Tags eingefasst.
- und werden hierarchisch ineinander gesetzt

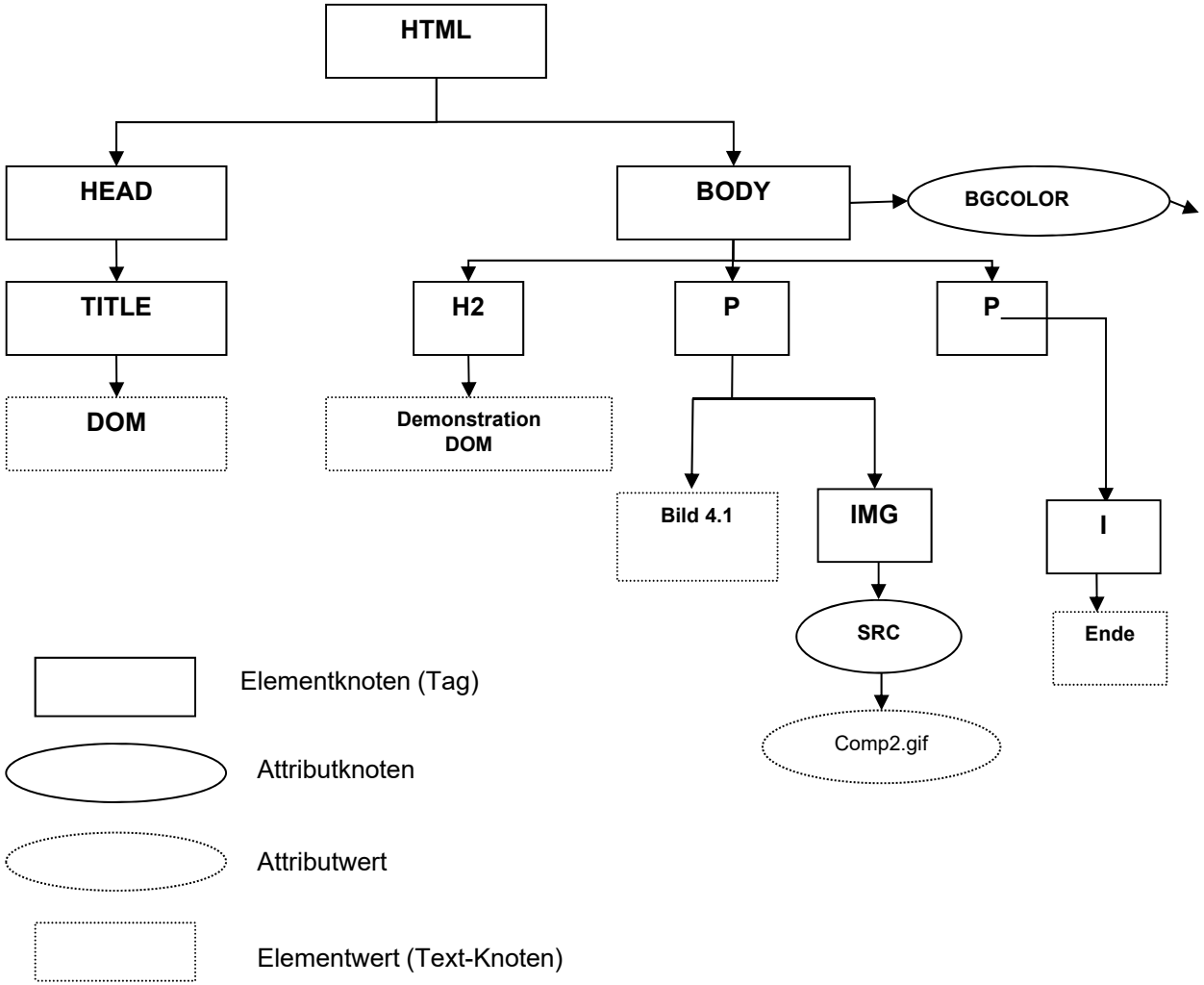
Darstellung als Baum mit Elementen auf gleicher Ebene und unter/übergeordneten Elementen möglich.

→ DOM: Document Object Model eines HTML-Dokuments

DOM (1)

Element-Beziehungen im letzten Beispiel:

```
<HTML>
 <HEAD>
  <TITLE>DOM</TITLE></HEAD>
 <BODY BGCOLOR="yellow">
  <H2>Demonstration  DOM</H2>
  <P>
   <IMG SRC="Comp2.gif" />  Bild 4.1
  </P>
  <P><I>Ende</I>
  </P>
 </BODY>
</HTML>
```



XML (1)

XML (Extensible Markup Language) Meta-Sprache, um Auszeichnungssprachen für Dokumente zu erzeugen

- Textuell lesbares Speicherformat
- Tags, um Bedeutung von Daten zu kennzeichnen, z.B. *<Datum>19.12.1979</Datum>* , Schachtelung der Tag-Strukturen
- XML ist maschinenlesbar und transformierbar
- Ideen von XML bereits alt, jedoch jetzt erst mit Tool-Unterstützung (XML-Editoren, XML-Transformatoren)

XML-konforme Untermengen:

- (X)HTML: Nachbau von HTML durch XML
- Grafikformat SVG (Scalable Vector Graphics) und weitere

XML (2)

Beispiel “Vorlesung“:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Vorlesung SYSTEM "C:\usr\Vorlesung.dtd">
<Vorlesung>
   <Titel lang="de">Internettechnologien</Titel>
   <Titel lang="en">Internet Technology</Titel>
   <Termin Semester-Id="SS14">
     <Ort>S527</Ort>
     <Wochentag>Montag</Wochentag>
     <Uhrzeit>11:10</Uhrzeit>
   </Termin>
</Vorlesung>
```

Ein XML-Dokument ist ***wohlgeformt***, wenn es der grundlegenden Syntax mit öffnendem und schließendem Tag, Attributen in doppelten Ausführungszeichen und einer hierarchischen Ordnung entspricht.

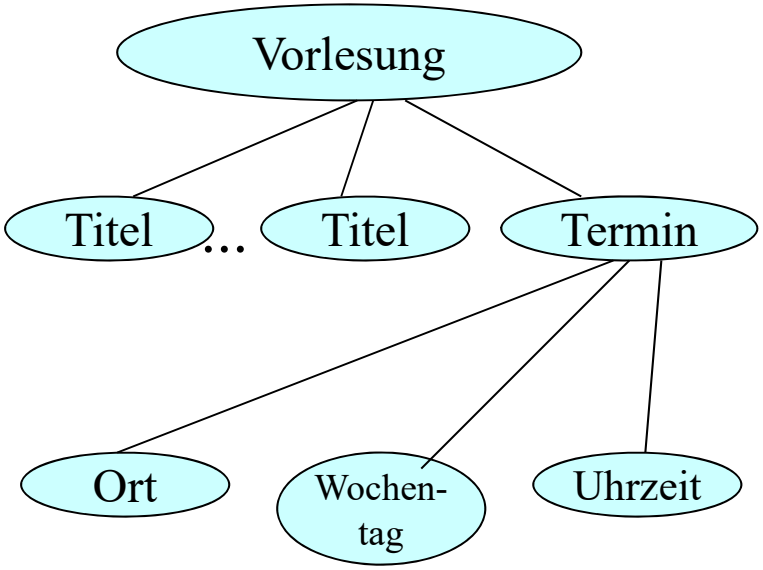
XML (3)

Die formale Struktur der Dokumente wird durch die DTD (Document Type Definition) bestimmt. XML Dokumente sind ***gültig*** (engl. valide), wenn sie dieser Struktur genügen. Gültigkeit setzt Wohlgeformtheit voraus.

DTD – Document Type Definition, Angabe im Prolog des XML-Dokuments (Verweis auf DTD-Datei oder URL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Vorlesung (Titel+,Termin)>
<!ELEMENT Titel (#PCDATA)>
<!ELEMENT Termin (Ort,Wochentag,Uhrzeit)>
<!ELEMENT Ort (#PCDATA)>
<!ELEMENT Wochentag (#PCDATA)>
<!ELEMENT Uhrzeit (#PCDATA)>

<!ATTLIST Titel
        lang  ID #REQUIRED
>
<!ATTLIST Termin
        Semester-Id ID #REQUIRED
>
```



XML (4)

Andere Variante zur Beschreibung der Struktur eines XML-Dokuments: **XML-Schema als Ersatz für DTD**

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ... >
<xs:element name="vorlesung"  type="VorlesungType">
<xs:annotation>
      <xs:documentation>Top Level Element</xs:documentation>
</xs:annotation>
</xs:element>
<xs:complexType name="VorlesungType">
      <xs:sequence >
      <xs:element name="Titel" type="TitelTyp" minOccurs="1"
                  maxOccurs="unbounded"/>
      <xs:element name="Termin" type="TerminTyp"/>
      </xs:sequence>
</xs:complexType>
```

Beispiel Fortsetzung:

```
<xs:complexType name="TitelTyp">
      <xs:sequence >
      <xs:element name="Titel" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="lang" type="xs:string"/>
</xs:complexType>
<xs:complexType name="TerminTyp">
     <xs:sequence >
      <xs:element name="Ort" type="xs:string"/>
      <xs:element name="Wochentag" type="xs:string"/>
      <xs:element name="Uhrzeit" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="Semester-Id" type="xs:string"/>
</xs:complexType>
</xs:schema>
```

Mehr zu XML-Schema folgt später.

XML-Beispiel “Musiksammlung“

Ein XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE musiksammlung SYSTEM "musiksammlung.dtd">
<musiksammlung eigentuemer="Max Meier">
  <album>
    <interpret>Kraftwerk</interpret>
    <albumtitel>Electric Cafe</albumtitel>
    <songs>     <song>Boing Boom Tschak</song>
     <song>Techno Pop</song> <song>Musique Non-Stop</song>
    <song>Der Telefon-Anruf</song>  <song>Sex Objekt</song>
    <song>Electric Cafe</song> </songs>
  </album>
  <album>
    <saenger>Helge Schneider</saenger>
    <albumtitel>Guten Tach</albumtitel>
    <songs >
      <song>Ansage </song>
      <song>Ladiladiho</song>
      <song>Der viereckige Hai</song>
    </songs>
  </album>
</musiksammlung>
```

Das Dokument zeigt deutlich den hierarchischen Aufbau ausgehend vom Wurzel-Element MUSIKSAMMLUNG und den vorhandenen Subelementen.

XML-Beispiel “Musiksammlung“ : DTD

DTD (Document Type Desciption) für das Beispiel „Musiksammlung“:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT musiksammlung (album)*>
<!ATTLIST musiksammlung
  eigentuemer CDATA #REQUIRED
>
<!ELEMENT album ((interpret|saenger|saengerin|band), albumtitel, songs) >
<!ELEMENT songs (song)* >
<!ELEMENT interpret (#PCDATA) >
<!ELEMENT saenger (#PCDATA) >
<!ELEMENT saengerin (#PCDATA) >
<!ELEMENT band (#PCDATA) >
<!ELEMENT albumtitel (#PCDATA) >
<!ELEMENT song (#PCDATA) >
```

XML Daten – Formatierte Anzeige im Browser (1)

für das Beispiel „Musiksammlung“:

Firefox

Internet-Technologien Test | file:///C:/Users/i...mlung_ohnecss.xml

file:///C:/Users/icasi/Desktop/Geomanagement/XML-Daten/musiksar

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft. Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

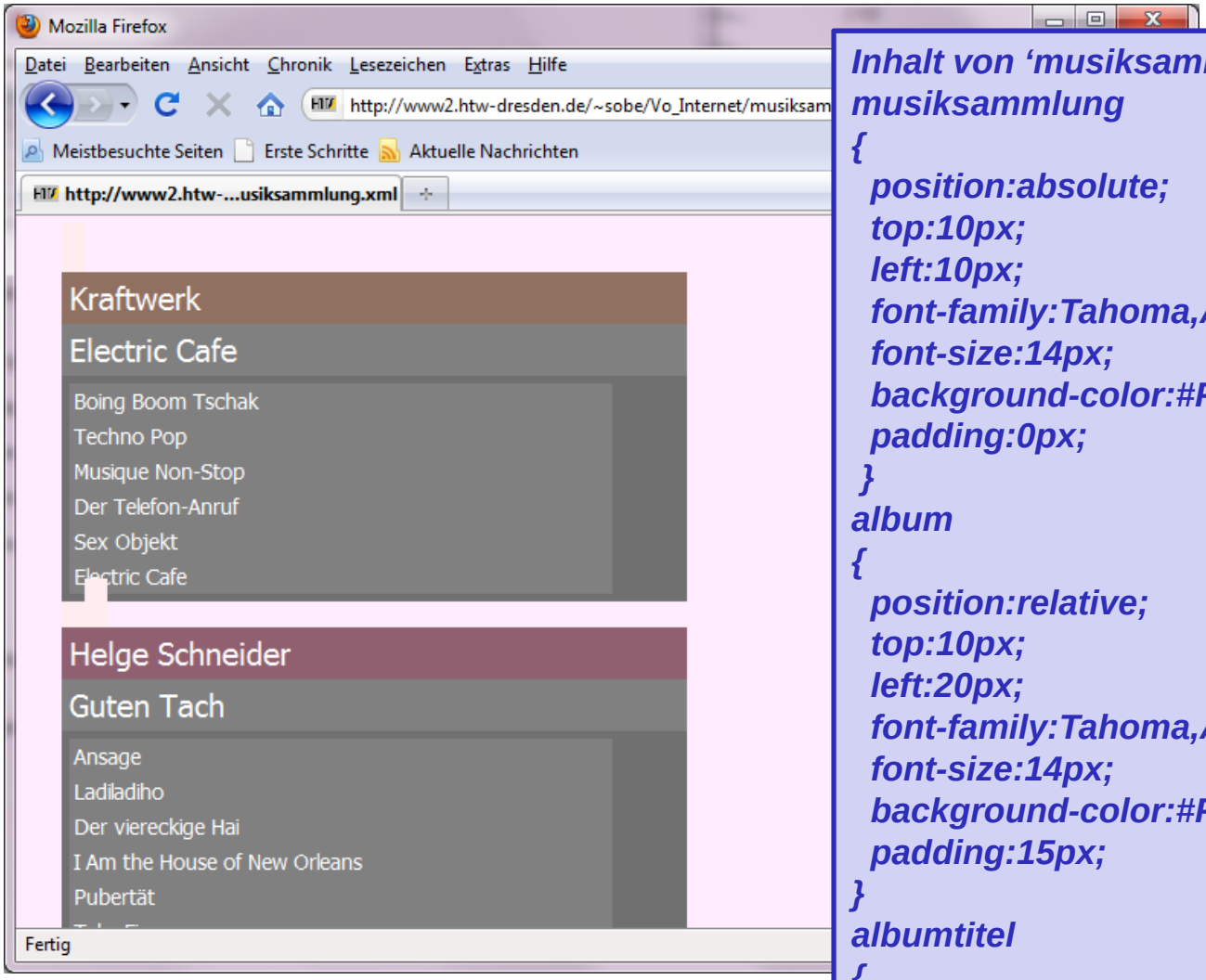
```
- <musiksammlung eigentuemer="Max Meier">
  - <album>
      <interpret>Kraftwerk</interpret>
      <albumtitel>Electric Cafe</albumtitel>
    - <songs>
        <song>Boing Boom Tschak</song>
        <song>Techno Pop</song>
        <song>Musique Non-Stop</song>
        <song>Der Telefon-Anruf</song>
        <song>Sex Objekt</song>
        <song>Electric Cafe</song>
      </songs>
    </album>
  - <album>
      <saenger>Helge Schneider</saenger>
      <albumtitel>Guten Tach</albumtitel>
```

Browser zeigen typischerweise die Struktur des XML-Dokuments durch Einrückungen an!

Eine grafisch formatierte Darstellung ist durch CSS Formatierung möglich.

XML Daten – Formatierte Anzeige im Browser (2)

für das Beispiel „Musiksammlung“ mit CSS-Formatierung:

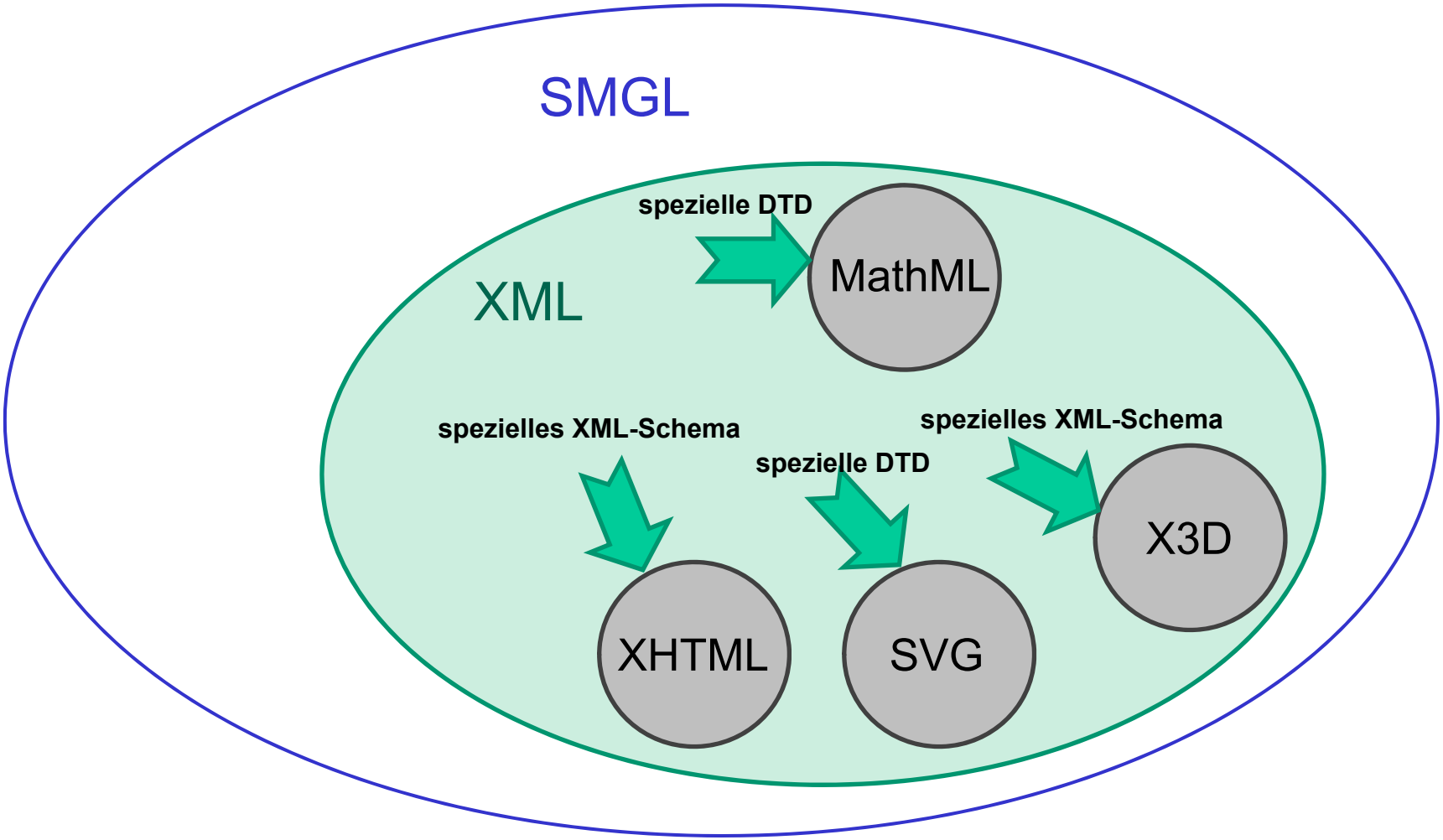


Inhalt von ‘musiksammlung.css‘

```
musiksammlung
{
  position:absolute;
  top:10px;
  left:10px;
  font-family:Tahoma,Arial,Helvetica,sans-serif;
  font-size:14px;
  background-color:#FFECFF;
  padding:0px;
 }
album
{
  position:relative;
  top:10px;
  left:20px;
  font-family:Tahoma,Arial,Helvetica,sans-serif;
  font-size:14px;
  background-color:#FFECEF;
  padding:15px;
}
albumtitel
{ ....
```

XML-Sprachen (1)

Genereller Ansatz:



XML-Sprachen (2)

Dokumente müssen XML-konform sein (Wohlgeformtheit), und zusätzlich die Einschränkungen und Regeln des speziellen Dokumentformats einhalten (Gültigkeit)

XML-Sprachen (4)

Durch die ständige Weiterentwicklung der XML-Technologien sind bereits für viele verschiedene Anwendungsgebiete neue wichtige XML-Sprachen definiert worden.

- Voice Extensible Markup Language (***VoiceXML***): Auszeichnungssprache für akustische Ausgabeplattformen.
- Mathematical Markup Language (***MathML***): Darstellung mathematischer Ausdrücke, Strukturierung des Dokumentes
- Die Synchronized Multimedia Integration Language (***SMIL***) Vokabular für Multimediapräsentationen im WWW.
- Scalable Vector Graphics (***SVG***): Vokabular für die Präsentation von zweidimensionalen Vektorgraphiken, in die auch Rastergraphiken eingebunden werden können.

XML-Sprachen (5)

Fortsetzung:

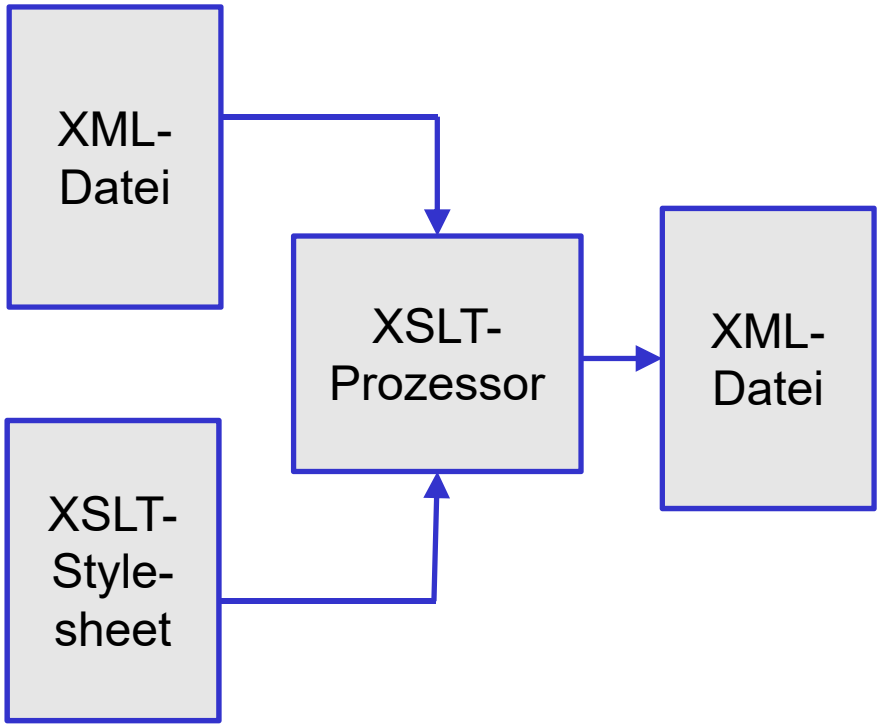
- XML-Schema (***XS***): anstelle einer DTD wird eine XML-basierte Beschreibung der Dokumentstruktur einer XML-Syntax beschrieben (siehe Beispiel zu Musiksammlung)

- Geography Markup Language (***GML***): zum Austausch raumbezogener Objekte im Bereich der Geodaten

- ***WSDL, SOAP***: XML-basierte Beschreibungen und Schnittstellen; werden für Webservices benutzt

XML-Sprachen (6)

XSLT ist eine Sprache, mit der beschrieben wird, wie ein XML-Dokument in ein anderes XML-Dokument umgewandelt werden soll.



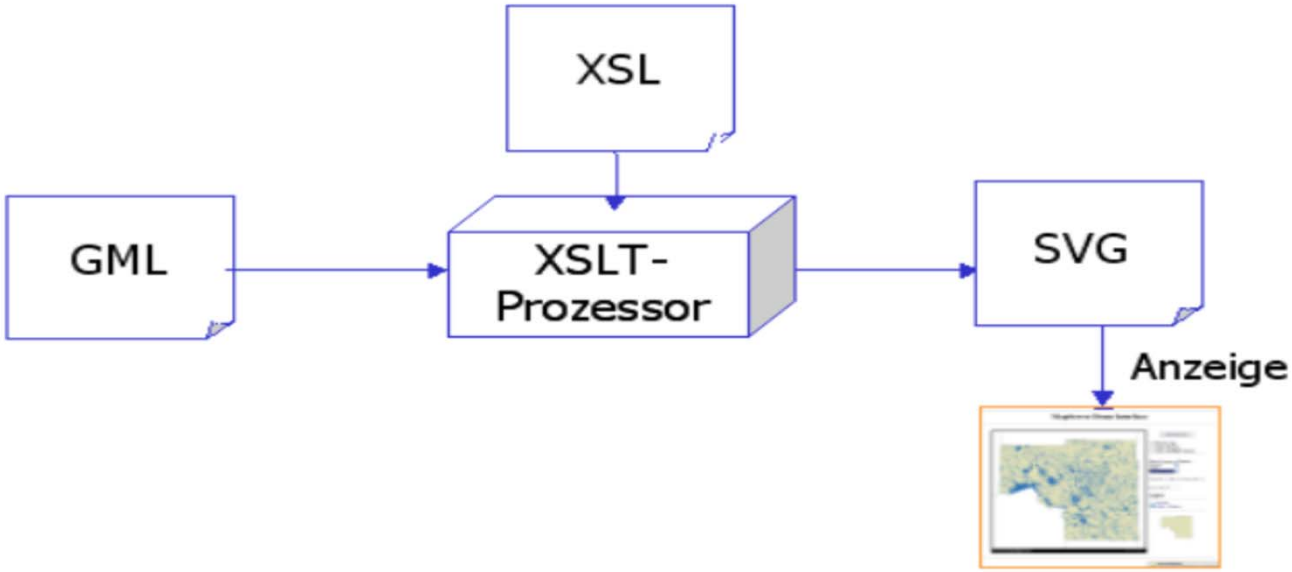
XML-Quelldatei und ein XSLT-Stylesheet (Regeln für die Transformation)

Das Programm, das die XSLT-Steueranweisungen versteht wird XSLT-Prozessor genannt

XML-Sprachen (7)

Transformation zwischen verschiedenen XML-Sprachen, zum Beispiel

- Dynamische Erzeugung von Webseiten
- Erzeugung von Grafiken (SVG)

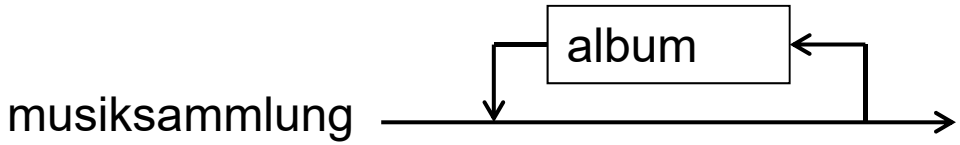


XML-Schema (1)

Ein XML-Schema kann anstelle einer DTD benutzt werden und ist selbst ein XML-Dokument.

Beispiel zur Musiksammlung:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema … xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="musiksammlung">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="album" />
      </xs:sequence>
      <xs:attribute name="eigentuemer" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
```

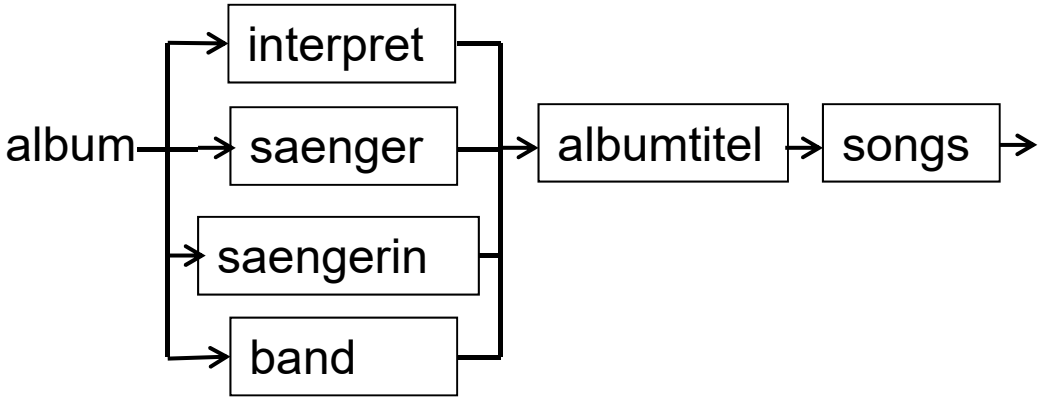


Dieser Teil beschreibt, dass eine Musiksammlung aus mehreren Alben (0 bis unbegrenzt) besteht.

XML-Schema (2)

Beispiel zur Musiksammlung (Fortsetzung):

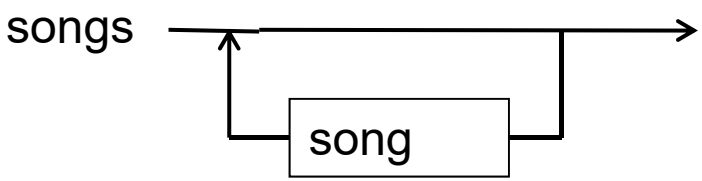
```
<xs:element name="album">
   <xs:complexType>
     <xs:sequence>
       <xs:choice>
         <xs:element ref="interpret" />  <xs:element ref="saenger" />
         <xs:element ref="saengerin" /> <xs:element ref="band" />
       </xs:choice>
       <xs:element ref="albumtitel" />
       <xs:element ref="songs" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
…
```



XML-Schema (3)

Beispiel zur Musiksammlung (Fortsetzung):

```
<xs:element name="songs">
   <xs:complexType>
     <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="song" />
     </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="interpret" type="xs:string" />
  <xs:element name="saenger" type="xs:string" />
  <xs:element name="saengerin" type="xs:string" />
  <xs:element name="band" type="xs:string" />
  <xs:element name="albumtitel" type="xs:string" />
  <xs:element name="song" type="xs:string" />
</xs:schema>
```



SVG (1)

SVG (Scalable Vector Graphics) ist ein offener Grafikstandard, der es erlaubt, zweidimensionale Grafikobjekte mit Hilfe von XML zu notieren (W3C 2003a)

SVG ist XML-basiert und unterstützt Vektordaten und zusätzlich die Einbettung von Rasterbildern, Audio- und Videodaten.

Eigenschaften:

- Auszeichnung für zweidimensionale Geometrieobjekte (Elemente mit geometriespezifischen Attributen)
- Operationen wie Transformationen, Gruppierung, Clipping und Filterung
- Formatierung der Elemente über CSS (Cascading Style Sheets)
- Ereignisbehandlung über Skripte mit Zugriff auf DOM.

SVG (2)

Weitere Eigenschaften:

- SVG-Grafiken sind im Browser skalierbar, ohne Qualitätsverlust (vgl. Bitmap-Grafiken)
- Interaktionen über Attribute, z.B. Einblendzeiten von Grafikobjekten
- Attribute können mit JavaScript gesetzt werden.
  Beispiel: Attribut visible="true" oder "false" zeitgesteuert setzen
- SVG kann in XHTML eingebettet werden. Über DOM kann dann z.B. ein JavaScript über das Document die Elemente der SVG Grafik adressieren. Damit können SVG-Grafiken durch clientseitiges Scripting dynamisch verändert werden

SVG (3)

Eine SVG-Grafik ist ein XML-Dokument, dessen Syntax gemäß eines XML-Schemas www.w3.org/2000/svg aufgebaut ist.

Struktur:

```
<svg xmlns="http://www.w3.org/2000/svg"  width="444" height="666">
<title>Ein Bild</title>
<desc>Hier eine Beschreibung zum Bild</desc>
<!-- hier folgen die Grafikelemente -->
</svg>
```

Ein SVG-Bild kann als externe Referenz in eine Webseite aufgenommen werden, oder auch in die Webseite eingebettet werden.

SVG (4)

Elemente beschreiben grafische Primitive, wie z.B.

- Linie: *<line x1="90" y1="150" x2="310" y2="150" style="stroke:black; stroke-width:2px;" />*
- Kreis: *<circle cx="50" cy="50" r="15" fill="yellow" stroke="black" />*
- Ellipse: *<ellipse cx="200" cy="50" rx="20" ry="40" style="fill:green;" />*
- Rechteck: *<rect x="0" y="0" width="100" height="100" style="fill:blue;" />*
- Text: *<text x="100" y="50">Guten Morgen</text>*

Für die Attribute sind verschiedene Maßeinheiten möglich:
px – Pixel (default), cm – Zentimeter, mm – Millimeter, …

SVG (5)

Elemente (Fortsetzung):

Polyline und Polygon:

```
<polyline fill="lightgray" stroke="red" stroke-width=„2px"
  points=" 176 10, 26 160, 326 160, 176 10" />
<polygon fill="darkblue" points="176 10, 26 160, 326 160" />
```

…beide Elemente beschreiben ein Dreieck

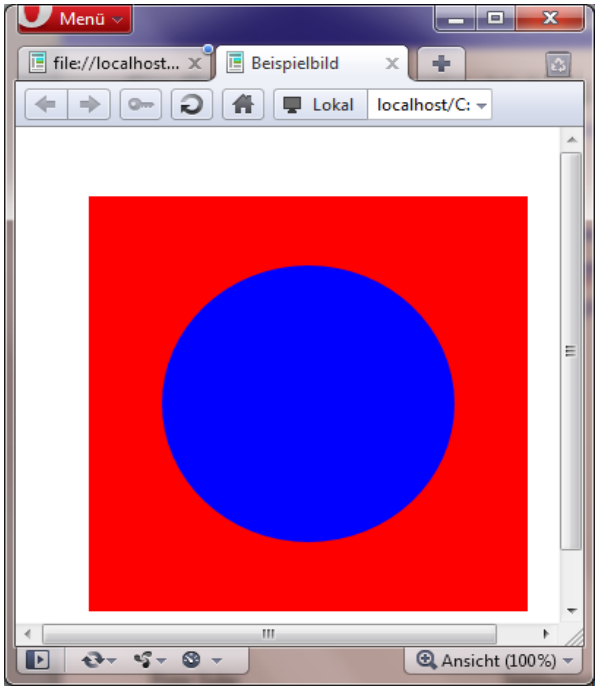
Verschachtelung:

```
<svg …>
<rect x="0" y="0" width="200" height="200" style="fill:red;" />
 <svg x=„20" y=„20" width="160" height="160">
   <rect x="0" y="0" width="100" height="100" style="fill:blue;" />
  </svg>
</svg>
```

SVG (6)

Ein Beispiel mit Verschachtelung:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="400">
<title>Beispielbild</title>
<desc>Ein einfaches SVG Bild</desc>
<rect x="50" y="50" width="300" height="300" style="fill:red;" />
  <svg x="50" y="50" width="300" height="300">
    <circle cx="150" cy="150" r="100"
      style="fill:blue;" />
  </svg>
</svg>
```



SVG (7)

SVG wird durch eine Reihe Vektorzeichenprogramme unterstützt, d.h. zum Erstellen von Grafiken ist primär keine Kenntnis des XML-Formats nötig.

SVG-XML-Struktur dann wichtig

- wenn grafische Inhalte automatisiert erzeugt werden sollen, z.B. bei der Programmierung einer SVG-Ausgabe
- Wenn auf SVG-Inhalte durch Skripte zugegriffen werden soll
- Bei der Transformation von anderen XML-Daten in SVG, z.B. Anzeige von geometrischen Strukturen

MathML

Durch MathML wird das äußere Erscheinungsbild mathematischer Ausdrücke kodiert (presentation).

Zusätzlich kann die Bedeutung eines Ausdrucks kodiert werden (content).

Beispiel: $(x+2)^3$

Presentation markup:

```
<math>
<msup>
  <mrow>
  <mo>(</mo>
   <mrow>
     <mi>x</mi> <mo>+</mo><mn>2</mn>
   </mrow>
  <mo>)</mo>
 </mrow>
 <mn>3</mn>
</msup>
</math>
```

Content markup:

```
<math>
 <apply>
  <power/>
  <apply>
   <plus/>
   <ci>x</ci>
   <cn>2</cn>
  </apply>
  <cn>3</cn>
  </apply>
</math>
```

XML-Eingabe und -Ausgabe mit Visual Basic .NET

Ein- und Ausgabe durch .NET-Klassen

XMLTextReader

und

XMLTextWriter

Diese Klassen unterstützen den Programmierer mit verschiedenen Methoden zu Lesen/Schreiben

- von XML-Tags mit eingeschlossenen Inhalten
- von Attributen

XML-Eingabe und -Ausgabe mit Visual Basic .NET

Beispiel zum Schreiben mit XMLTextWriter

```
Dim xmlwr As XmlTextWriter
xmlwr = New XmlTextWriter(xmlFilename, New UnicodeEncoding)

xmlwr.WriteStartElement( "a" )
xmlwr.WriteAttributeString("x", "5")

xmlwr.WriteStartElement("b" )
xmlwr.WriteString("ccc")
xmlwr.WriteEndElement()

xmlwr.WriteEndElement()
```

Erzeugt:

```
<?xml version="1.0"?>
<a x="5">
  <b>ccc</b>
</a>
```

XML-Eingabe und -Ausgabe mit Visual Basic .NET

Lesen mit XMLTextReader

```
Dim xmlr As XmlTextReader
xmlr = New XmlTextReader(Filename)
Do While xmlr.Read()
        If xmlr.NodeType = XmlNodeType.Element Then
              Console.WriteLine("Node-Name:" + xmlr.Name)
              If xmlr.AttributeCount > 0 Then
                Do While xmlr.MoveToNextAttribute()
                    Console.WriteLine("Attr-Name:" + xmlr.Name + " Attr-Value:" + xmlr.Value)
                Loop
              End If
       ElseIf xmlr.NodeType = XmlNodeType.Text Then
              Console.WriteLine("Node-Value:" + Format(xmlr.Value))
       End If
Loop
```

Liest Knoten (Tags und Attribute) in der Reihenfolge ihres Auftretens in der Datei. Für Beispiel auf vorheriger Seite würde die folgende Ausgabe erzeugt:

```
Node-Name: a
Attr-Name: x Attr-Value: 5
Node-Name: b
Node-Value: ccc
```

Exkurs: Web-Nutzerschnittstellen mit Active Server Pages ASP.NET

Allgemeines:

ASP ist die serverseitige Schnittstelle zu Skripten/Programmen ausgehend vom Microsoft-Internet-Information-Server
(z.B. in Windows-Server enthalten)

Skript- und Programmiersprachen
ASP 2/3: Visual Basic Skript und JSkript

ASP.NET:
mehrere Sprachen im Rahmen von .NET:

- Visual Basic
- C/C++
- C#

ASP.NET

Konzept/Varianten:

- **Single-File-Modell:** eingebetteter Code (Inline, durch <% … %> eingefasst). Mischung von Webseitencode in HTML mit Programmcode (Visual Basic, C# oder andere .NET-Sprachen)

- **Code-Behind-Modell:** gesonderte Quelltextmodule mit Klassen, die Methoden für Web-Elemente bereitstellen

Durch das ASP-Framework werden Elemente zur clientseitigen Interaktion mittels JavaScript automatisch generiert

Der Programmcode wird übersetzt und innerhalb des IIS (Microsoft Webserver) ausgeführt.

Entwicklungsumgebung: Microsoft Visual Studio
mit Visual Web Developer (lokaler Webserver)

ASP: eingebetteter Visual Basic Code

Beispiel für eine ASP.NET- Webseite:

```
<%@ Page Title="Beispiel" Language="vb" MasterPageFile="~/Site.Master"
  AutoEventWireup="false“ CodeBehind="Default.aspx.vb„
  Inherits="WebAppASPNET._Default" %>
 <asp:Content ID="HeaderContent" runat="server„
  ContentPlaceHolderID="HeadContent">
 </asp:Content>
 <asp:Content ID="BodyContent" runat="server"
  ContentPlaceHolderID="MainContent">
   <h2> Eine ASP.NET-Beispielanwendung </h2>
   <%      For i = 1 To 3  %>
   <p> Hallo </p>
   <% Response.Write(i)
        Next
   %>
  </asp:Content>
```

HTML-, Head- und Body-Tags werden serverseitig automatisch erzeugt.

gewöhnliche HTML- Elemente

eingebetteter Code (hier Visual Basic)

ASP – Beispiel für eine Code-Behind-Anwendung

Webseite:

```
<%@ Master Language="VB" AutoEventWireup="false" CodeBehind="Site.master.vb"
Inherits="Eisladen.Site" %>

…
<form id = "form1" runat="server">
 <p>Dein Name:<input NAME="personname" SIZE="20"/></p>
 <p>Dein Alter:<input NAME="alter" SIZE="3"/></p>
 <p>Deine Eis-Bestellung:
 <SELECT NAME="bestellung">
        <OPTION>Schoko</OPTION>
        <OPTION>Erdbeere</OPTION>
        <OPTION>Waldfrucht</OPTION>
         <OPTION>Eierlikoer</OPTION>
          <OPTION>Rum-Trauben</OPTION>
  </SELECT>
  </p>
   <p>
 <asp:Button ID="Auswerten_Button" runat="server" OnClick="Form_Processing"
                Text = "Bestellung absenden"> </asp:Button>
   </p>
 </form>
```

ASP – Beispiel für eine Code-Behind-Anwendung

Visual Basic Code (siteMaster.vb) :

```
Public Class Site
   Inherits System.Web.UI.MasterPage

   Protected Sub Form_Processing(ByVal sender As Object,
                 ByVal e As System.EventArgs) Handles Auswerten_Button.Click

      Label1.Text = "Bestellung eingegangen:" & DateTime.Now.ToString()
      Dim besteller As String = Request.Form("personname")
      Dim alter As Integer = Integer.Parse(Request.Form("alter"))
      Dim sorte As String = Request.Form("bestellung")

      Label2.Text = "Bestellung von: " & besteller & "(" & Format(alter) & ")"

      Label3.Text = "Bestellte Sorte: " & sorte

      If (alter < 18 AND (sorte = "Eierlikoer" OR sorte = "Rum-Trauben")) Then
         Label4.Text = "Bestellung abgelehnt, wegen Alkohol."
      Else
         Label4.Text = "Bestellung OK"
      End If
   End Sub
End Class
```

ASP – Beispiele für Single-File-Anwendungen (1)

Beispiele aus http://www.w3schools.com/

Formular-Anwendungen mit GET-Operation und Zugriff auf Query-String:

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_simplereqquery.asp" method="get">
   First name: <input type="text" name="fname"><br>
   Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>

<% Response.Write(Request.QueryString) %>
</body>
</html>
```

Webseite ruft sich per Formular-Submit hier selbst wieder auf

Quelle:
http://www.w3schools.com/asp/showasp.asp?filename=demo_simplereqquery

ASP – Beispiele für Single-File-Anwendungen (2)

GET-Methode und Zugriff auf Query-Parameter

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_reqquery.asp" method="get">
  Your name: <input type="text" name="fname" size="20" />
  <input type="submit" value="Submit" />
</form>

<% dim fname
     fname=Request.QueryString("fname")
     If fname<>"" Then
       Response.Write("Hello " & fname & "!<br>")
       Response.Write("How are you today?")
    End If
%>
</body>
</html>
```

Auch diese Webseite ruft sich per Formular-Submit selbst wieder auf

Quelle: http://www.w3schools.com/asp/showasp.asp?filename=demo_reqquery

ASP – Beispiele für Single-File-Anwendungen (3)

POST-Methode und Nutzung der Request- und Response-Objekte

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_simpleform.asp" method="post">
  Your name: <input type="text" name="fname" size="20" />
  <input type="submit" value="Submit" />
</form>
<% dim fname
     fname=Request.Form("fname")
     If fname<>"" Then
        Response.Write("Hello " & fname & "!<br>")
        Response.Write("How are you today?")
     End If
%>
</body>
</html>
```

Auch diese Webseite ruft sich per Formular-Submit selbst wieder auf

Quelle:http://www.w3schools.com/asp/showasp.asp?filename=demo_simpleform

ASP.NET: Nutzer-Interaktion

Das **Request-Objekt** stellt die notwendigen Daten und Methoden bereit, um Eingaben vom Client auszuwerten.

Das Request-Objekt besteht aus (Auflistung nicht vollständig):

- **Form-Array**: um Formulardaten zu lesen, die mit der Methode POST abgesendet wurden
- **Query-String**: für mit der GET-Methode abgesendete Formulardaten
- **ServerVariables-Array**: für Daten aus dem HTTP-Header
- **Cookies-Array**: um Cookies auszulesen
- **TotalBytes- und BinaryRead-Properties**:
  um Datei-Uploads zu behandeln

ASP.NET – Request-Objekt und Form-Array

Das **Form-Array** enthält die von einem Formular (FORM-Tag in HTML) gelesenen Daten, bei der Übertragung mit der POST-Methode (Attribut method="post").

Attribute:

REQUEST.Form.Count … Anzahl der Eingabefelder in einem Formular

REQUEST.Form("name") … Zugriff auf ein Eingabeelement mit den name-Attribut wie angegeben

Möglicherweise können Felder im Formular mehrere Werte besitzen. Dann ist *REQUEST.Form("name")* selbst ein Feld, das wie folgt ausgewertet werden kann:

REQUEST.Form.Count ("name") … Anzahl der Werte, die das Eingabeelement "name" erzeugt hat

REQUEST.Form("name")(index) … Zugriff auf den durch *index* indizierten Wert

ASP.NET – Request-Objekt / QueryString-Array

Das QueryString-Array stellt einem Skript die durch die GET-Methode übertragenen Formulardaten bereit (FORM-Attribut method="get").

REQUEST.QueryString.Count … Anzahl der Forularfelder

REQUEST.QueryString("name") … Wert eines durch name identifizierten Formularfeldes

Möglicherweise können Felder im Formular mehrere Werte besitzen. Dann ist REQUEST.QueryString ("name") selbst ein Feld, das wie folgt ausgewertet werden kann:

REQUEST.QueryString.Count ("name") … erzeugt die Anzahl der Werte in dem durch Name identifizierten Eingabeelement

REQUEST.QueryString("name")(index) … zum Zugriff auf den durch *index* spezifizierten Wert des Eingabeelement mit *name*

ASP.NET – Request-Objekt /Datei-Upload

Das Request-Objekt kann bei einem Datei-Upload die Größe der angegebenen Datei und deren Inhalt zurückliefern.

Der Upload muss mit der Methode method=POST erfolgen.

REQUEST.TotalBytes … liefert de Anzahl an Bytes in der Datei (including header information)

REQUEST. BinaryRead(1) … realisiert den Zugriff auf die Datei

Die beiden Elemente können aber nicht zusammen mit den restliche Formulardaten übertragen und abgefragt werden. Deshalb muss das Form-Tag den zusätzlichen Parameter enthalten:

enctype="multipart/form-data"

ASP.NET – Server Variablen

ServerVariables-Array

Das Request-Objekt enthält die ServerVariables mit Informationen über den HTTP-Server, den Client und den Request.

Beispiele für Server-Variablen:

HTTP_USER_AGENT (Browserkennung), CONTENT_TYPE, CONTENT_LENGTH

Hier ein Skript, das alle Servervariablen auflistet:

```
<table border=0> <%
    for each var in Request.ServerVariables
            content = Request.ServerVariables(var)
             if content = "" then content = "<b>(undefined)</b>“ %><TR>
             <TD class="varname"><%=var%></TD>
             <TD class="varcontent"><%=content%></TD></TR>
   <% next %>
</table>
```

ASP.NET – Cookies

Das Request– und das Response-Objekt erlauben, s.g. Cookies auf dem Client zu lesen und zu schreiben. Dazu gibt es ein Cookie-Array. Die Cookies können z.B. einen Nutzer wiedererkennen, oder einen Warenkorb speichern.

Setzen eines Cookies beim Client:

RESPONSE.Cookies("cookie-name") = "cookie-value"

RESPONSE.Cookies("cookie-name").expires =CDate("14.5.2012")

Das Löschen eines Cookies erfolgt indirekt über das Setzen einer Gültigkeitsobergrenze auf ein Datum in der Vergangenheit:

RESPONSE.Cookies("cookie-name").expires = CDate("14.2.2012")

Lesen eines Cookies über einen Request :

content= REQUEST.Cookies("cookie-name")