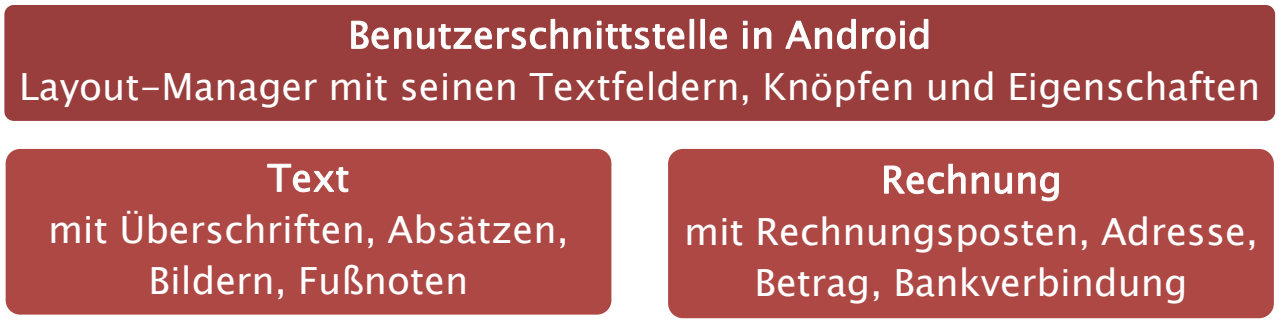
XML: Einführung in die eXtensible Markup Language (XML)

- Verstehen welchen Zweck XML verfolgt
- Aufbau einer XML-Datei verstehen und mit XML-Elementen umgehen können
- Wissen wie die Struktur von XML-Dokumenten definiert werden kann
- Die verschiedenen Möglichkeiten der Definition kennen
- Mit Namensräumen umgehen können
- Die Möglichkeiten kennen die Java bietet, um mit XML-Dateien umzugehen
- Mit StAX einen einfachen Parser für XML-Dateien schreiben können

eXtensible Markup Language (XML) ist eine *Auszeichnungssprache* welche zur strukturierten Beschreibung hierarchisch organisierter Daten in Form von *Textdateien* dient und insbesondere zum *Datenaustausch* zwischen Computern verwendet wird[1]



[1] Vom World Wide Web Consortium (W3C) standardisiert

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<party datum="31.12.14">
  <gast name="Albert Angsthase">
    <getraenk>Wein</getraenk>
    <getraenk>Bier</getraenk>
    <zustand ledig="true" nuechtern="false"/>
  </gast>
  <gast name="Elsa Ehrlich">
    <getraenk>Apfelsaft</getraenk>
    <zustand ledig="true" nuechtern="true"/>
  </gast>
</party>
```

- Dokument muss *Wurzelelement* (<party>)haben, das alle anderen einschließt
- *Element* (*engl. tag*) kann weitere Elemente (<gast>) oder Text (<getraenk>) enthalten
- Element kann Attribute (ledig="true") enthalten
- Attribut besteht aus *Attributname* und *Wert*
- *Wert* wird mit ' oder " eingeschlossen
- Elemente *mit* Inhalt (<getraenk>Wein</getraenk>)
- Elemente *ohne* Inhalt und (evtl.) Attribute (<zustand ledig="true"/>)
- Groß-/Kleinschreibung wichtig
- *Kommentare* mit <!-- Kommentar -->

Problem:

Wie wird ermittelt, ob XML-Dokument korrekt (engl. *valid*) ist?

- Werden die richtigen Elemente verwendet?
- Werden die Elemente richtig verschachtelt?
- Welche Elemente sind zwingend vorgesehen?
- Welche Attribute darf ein Element besitzen?

Lösung:

Document Type Definition (DTD) **XML Schema Definition (XSD)**

Beschreibungssprachen für den Aufbau von XML-Dokumenten

Beispiel DTD

Datei `party.dtd`

```
<!ELEMENT party (gast*)>
<!ATTLIST party datum CDATA #REQUIRED>
<!ELEMENT gast (getraenk*, zustand)>
<!ATTLIST gast name CDATA #REQUIRED>
<!ELEMENT getraenk (#PCDATA)>
<!ELEMENT zustand EMPTY>
<!ATTLIST zustand ledig CDATA #IMPLIED nuechtern CDATA #IMPLIED>
```

CDATA Characterdata
? <=1, + >=1, * >= 0
keine Angabe ==1
#IMPLIED kann
EMPTY keine Unterelemente

Bezugnahme auf eine DTD in party.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE party SYSTEM "party.dtd">
<party datum="31.12.2014">
  …
</party>
```

- Hinter DOCTYPE muss *Wurzelelement* angegeben werden
- Hinter SYSTEM steht *URI mit Adresse* der DTD-Datei

Anmerkungen für Eclipse

- XML-Dateien können bei bekannter DTD-Datei validiert werden
- Bei Erstellung einer XML-Datei schlägt Eclipse mögliche Tags mit ihren Eigenschaften vor

XML Schema Definition

- Da Strukturbeschreibung auch im XML-Format vorliegt, wird Parsen vereinfacht
- *Detailliertere Beschreibung* als in DTD möglich
- Übliche Datentypen wie `string`, `integer`, `double`, `date`, `duration` bereits vorhanden
- Definition *eigener Datentypen* möglich

Namensräume

Werden benötigt um Elemente zu Gruppen zusammenzufassen denn

- Elementnamen können mit *unterschiedlicher Bedeutung* in *mehreren Schemen* vorkommen
- Elemente unterschiedlicher Schemen können in ein und *derselben XML-Datei verwendet* werden

Datei `party.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<pty:party xmlns:pty="http://www.tfobz.net/partySchema"[1]
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.tfobz.net/partySchema party.xsd"[2]
  datum="31.12.14">
  <pty:gast name="Albert Angsthase">
    …
  </pty:gast>
</pty:party>
```

- *Namensraum* ist Verknüpfung von Präfix und URI
- Namensraum wird durch `xmlns` normalerweise an *Wurzelelement* gebunden[1]
- URI muss nicht unbedingt existieren
- Verbindet Schema mit der Schema-Datei[2]

Datei party.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"[1]
  xmlns="http://www.tfobz.net/partySchema"[2]
  targetNamespace="http://www.tfobz.net/partySchema"[3]
  elementFormDefault="qualified">[4]

  <xsd:element name="party" type="partyType"/>

  <xsd:complexType name="partyType">
    <xsd:sequence>
      <xsd:element name="gast" type="gastType"
        minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="datum" type="datumType" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="gastType">
    <xsd:sequence>
      <xsd:element name="getraenk" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="zustand" type="zustandType" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:simpleType name="datumType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-3][0-9].[0-1][0-9].[0-9][0-9]" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="zustandType">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:attribute name="nuechtern" type="xsd:boolean" />
        <xsd:attribute name="ledig" type="xsd:boolean" />
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

- Definiert den Namensraum für XML Schema Definition, so dass auch diese Datei validiert werden kann[1]
- Definiert den Namensraum für die in diesem Schema neu definierte Typen (gastType, datumType, zustandType)[2]
- Definiert die URI für das durch die Datei definierte Schema. URI muss in den XML-Dateien verwendet werden, um Validierung nach diesem Schema vornehmen zu können[3]
- elementFormDefault="qualified" in Kombination mit [2] legt fest, dass alle Elemente – auch die lokal in dieser Datei definierten – mit Präfix versehen werden müssen[4]

Java-APIs für XML

Baumbasierte APIs
Gesamter XML-Baum im Speicher verfügbar, Suchoperationen, Sortieren, interaktives Arbeiten möglich
(DOM Document Object Model)

Ereignisbasierte APIs
Parser liest XML-Datei blockweise ein, benötigt deshalb weniger Speicher und wirft Ereignis bei Elementvorkommen
(SAX Simple API for XML)

StAX (Streaming API for XML)
Mittelding obiger Verfahren welches nach dem Iterator-Prinzip die XML-Datei parsen lässt

Serielle Verarbeitung einer XML-Datei mit StAX

- Erzeuge eine `XMLInputFactory`
- Erzeuge `XMLStreamReader` für die Verarbeitung
- Erfrage mit `next()` die nächste Komponente der XML-Datei
- Ermittle Typ der Komponente und verarbeite sie

Einfacher StAX-Parser

```
XMLInputFactory factory = XMLInputFactory.newInstance();
InputStream in = new FileInputStream(XML_FILE);
XMLStreamReader parser = factory.createXMLStreamReader(in);

String characters = "";
while (parser.hasNext()) {
  int elementType = parser.next();
  switch (elementType) {
    case XMLStreamConstants.START_DOCUMENT: {
      System.out.println(elementType +
        " START_DOCUMENT: " + parser.getVersion());
      break;
    }

    case XMLStreamConstants.END_DOCUMENT: {
      System.out.println(elementType + " END_DOCUMENT: ");
      parser.close();
      break;
    }

    case XMLStreamConstants.NAMESPACE: {
      System.out.println(elementType +
        "NAMESPACE: " + parser.getNamespaceURI());
      break;
    }

    case XMLStreamConstants.START_ELEMENT: {
      characters = "";
      System.out.print(elementType + " START_ELEMENT: " +
        parser.getLocalName() + " ");
      for (int i = 0; i < parser.getAttributeCount(); i++)
        System.out.print(parser.getAttributeName(i) + "=" +
          parser.getAttributeValue(i) + " ");
      System.out.println();
      break;
    }

    case XMLStreamConstants.END_ELEMENT: {
      System.out.println(elementType + " END_ELEMENT: " +
        parser.getLocalName() + " " + characters;
      characters = "";
      break;
    }

    case XMLStreamConstants.CHARACTERS: {
      if (!parser.isWhiteSpace() && parser.getText() != null &&
        parser.getText().leng
        characters+=parser.ge
      break;
    }
  }
}
```

Console

<terminated> SampleParser [Java Application] G:\Informatik\Prograı

```
1 START_ELEMENT: party datum=31.12.2014
1 START_ELEMENT: gast name=Albert Angsthase
1 START_ELEMENT: getraenk
2 END_ELEMENT: getraenk Wein
1 START_ELEMENT: getraenk
2 END_ELEMENT: getraenk Bier
1 START_ELEMENT: zustand ledig=true nuechtern=false
2 END_ELEMENT: zustand
2 END_ELEMENT: gast
1 START_ELEMENT: gast name=Elsa Ehrlich
1 START_ELEMENT: getraenk
2 END_ELEMENT: getraenk Milch
1 START_ELEMENT: zustand ledig=true nüchtern=true
2 END_ELEMENT: zustand
2 END_ELEMENT: gast
2 END_ELEMENT: party
8 END_DOCUMENT:
```

JSON (JavaScript Object Notation) die Alternative zu XML

```
{
  "datum": "31.12.14",
  "gaeste": [
    {
      "name": "Albert Angsthase",
      "getraenke": ["Wein", "Bier"],
      "zustand": {
        "ledig": true,
        "nuechtern": false
      }
    },
    {
      "name": "Elsa Ehrlich",
      "getraenke": ["Apfelsaft"],
      "zustand": {
        "ledig": true,
        "nuechtern": true
      }
    }
  ]
}
```

- Syntax von JSON ist einfacher gestaltet, daher leichter les- und schreibbarer
- Weniger Overhead beim Transport über Netzwerk