Semistrukturierte Daten

Sommersemester 2010

Teil 5: Java API for XML Processing

5.1. Überblick

5.2. SAX (Simple API for XML)

5.3. DOM (Document Object Model)

5.4. Serialisierung von XML Daten

5.5. Epilog

5.1. Überblick

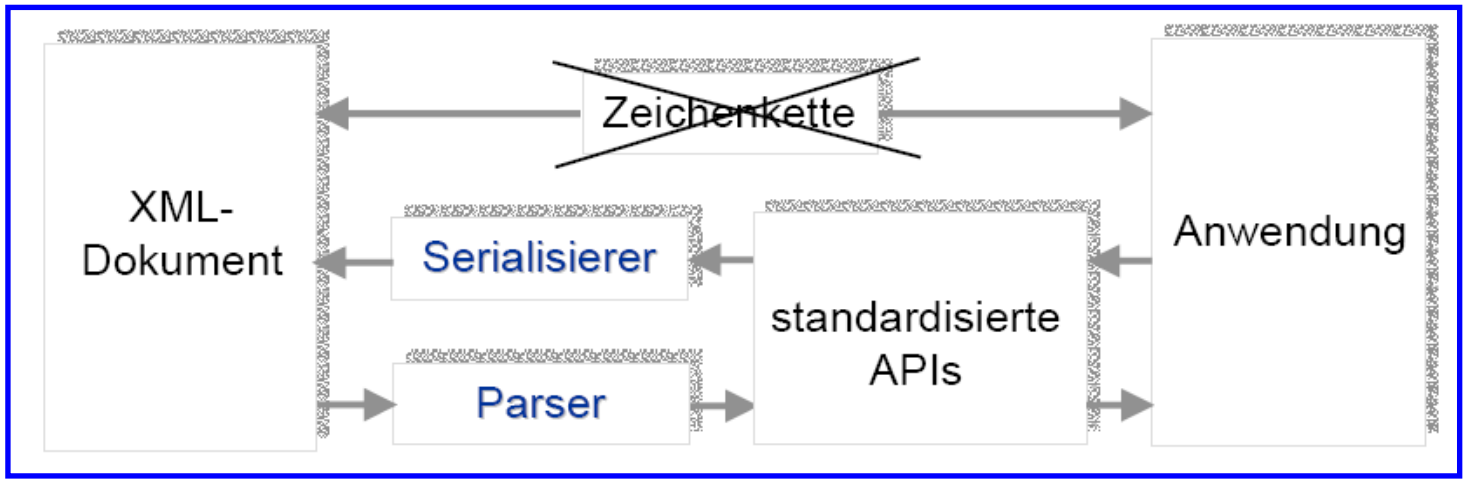
- XML-APIs und Java
- XML-Prozessor (Parser)
- Parser-Modelle

XML-APIs und Java

- Die wichtigsten XML-APIs (DOM und SAX) sind eigentlich programmiersprachen-unabhängig. In der SSD-Vorlesung werden aber nur die Java-Realisierungen behandelt.
- JAXP (Java API for XML-Processing):
  - Teil von JDK ab Version 1.4
  - Enthält Java-Realisierung von DOM und SAX sowie ein XSLT-API (TrAX: Transformation API for XML)
- Wichtige JAXP-Packages:
  - javax.xml.parsers (gemeinsames Interface für SAX und DOM Parser von unterschiedlichen Herstellern).
  - org.w3c.dom
  - org.xml.sax
  - javax.xml.transform

Idee eines XML-Prozessors (Parsers)

- Stellt der Applikation eine einfache Schnittstelle zum Zugriff auf ein XML-Dokument (plus eventuell zum Manipulieren und wieder Abspeichern des XML-Dokuments) zur Verfügung.
- Fokus auf die logische Struktur bzw. den Inhalt des XML-Dokuments (und nicht auf die exakte Syntax) => wesentlich flexibler/robuster als Text-Parsen.



Aufgaben eines XML-Prozessors (Parsers)

- Überprüfung der Wohlgeformtheit und (optional) der Gültigkeit
- Ergänzung von default/fixed-Werten
- Auflösen von internen/externen Entities und Character References
- Normalisierung von Whitespace

Parser-Modelle

- Objektmodell Parser:
  - Baut gesamten XML-Baum im Speicher auf => wahlfreier Zugriff
  - Beispiele: DOM, JDOM
- Push Parser (ereignisorientiert, Parser kontrolliert Ablauf):
  - XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
  - Applikation kann für die möglichen "Events" (d.h. syntaktische Einheiten) callback Funktionen bereitstellen
  - Beispiel: SAX
- Pull Parser (ereignisorientiert, Applikation kontrolliert Ablauf):
  - XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
  - Applikation fordert Analyse der nächsten syntaktischen Einheit an
  - Beispiel: StAX (mittlerweile Teil von Java als SJSXP, "Sun Java Streaming XML Parser")

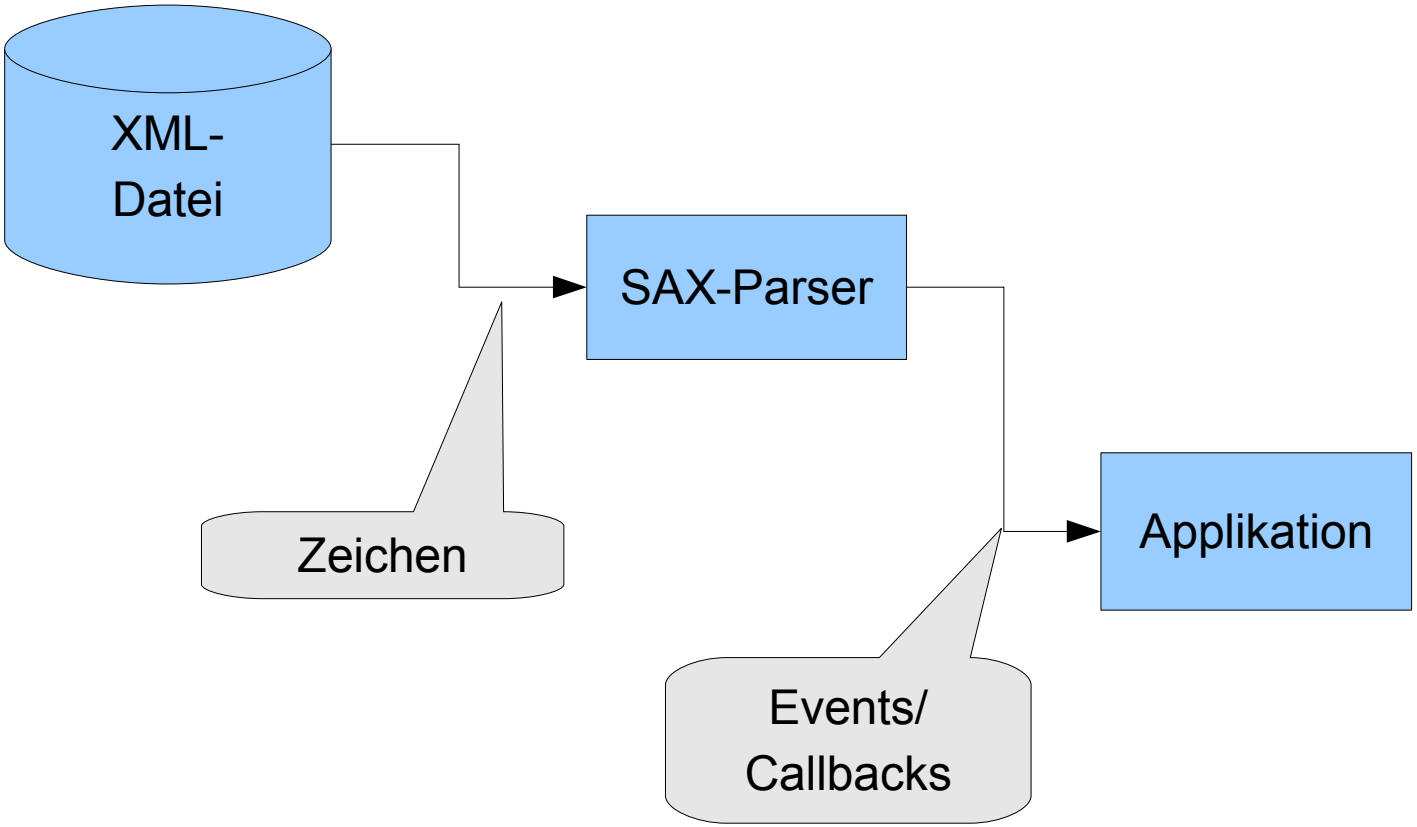
5.2. Simple API for XML (SAX)

- Entwicklung von SAX
- Funktionsweise von SAX
- ContentHandler
- Weitere Eventhandler
- XMLReader
- SAX Filter

Entwicklung von SAX

- SAX: Simple API for XML
- keine W3C Recommendation, aber ein de-facto Standard: http://www.saxproject.org/
- Free and Open Source Software (SourceForge)
- Plattform- und Programmiersprachen-unabhängig (auch wenn SAX ursprünglich für Java entwickelt wurde)
- SAX Versionen:
  - SAX 1.0: entstanden auf Initiative von Peter Murray-Rust (mit dem Ziel, mehrere ähnliche aber inkompatible Java XML-APIs zusammenzuführen), im Mai 1998 freigegeben.
  - SAX 2.0: erweitert um Namespace-Unterstützung; nicht kompatibel zu SAX 1.0 (Aktuell SAX 2.0.1)

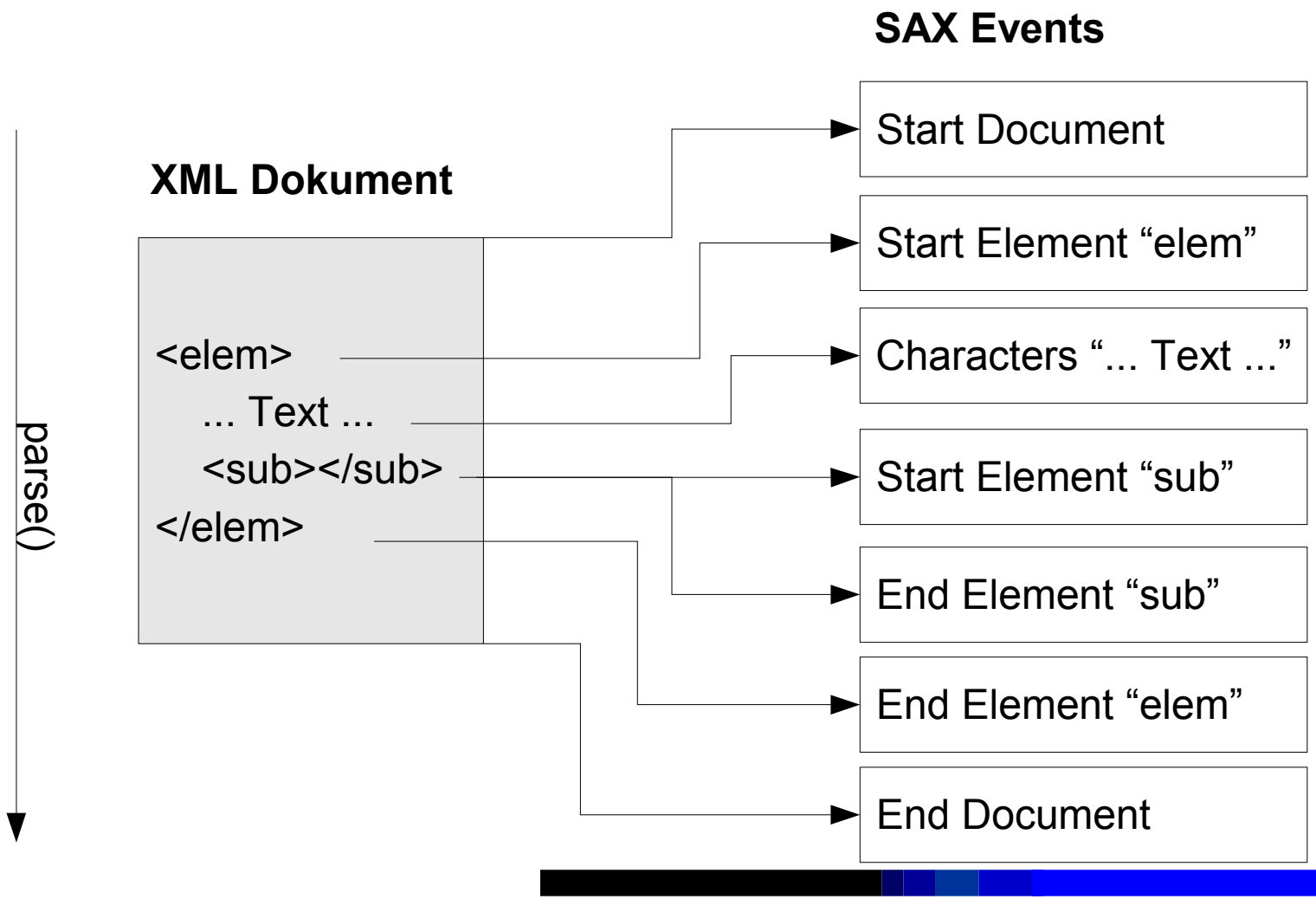
Funktionsweise von SAX



Funktionsweise von SAX

- Applikation registriert callback Funktionen beim SAX-Parser.
- Applikation stößt den Parser an.
- SAX-Parser durchläuft das XML-Dokument einmal sequentiell.
- Parser erkennt syntaktische Einheiten des XML-Dokuments.
- Parser ruft für jedes Event die entsprechende callback Funktion auf.
- Diese callback Funktionen sind auf 4 Event Handler aufgeteilt: ContentHandler, ErrorHandler, DTDHandler, EntityResolver

Einfaches Beispiel



XML Reader (Preview)

- Ist der eigentliche SAX-Parser, d.h.: liest das XML-Dokument und ruft die callback Funktionen auf.
- Erzeugen des XML Readers:
```
XMLReader xr = XMLReaderFactory.createXMLReader();
```
- Registrieren der callback Funktionen
```
xr.setContentHandler(ContentHandler ch);
...
```
- Methode zum Anstoßen des Parsers:
```
xr.parse(InputSource is);
```

ContentHandler

- Überblick
- einige ContentHandler-Methoden im Detail:
  - Dokument-Verarbeitung
  - Element-Verarbeitung
  - Attributes Interface
  - Text-Verarbeitung
  - Locator-Interface
- DefaultHandler

Überblick: Methoden des ContentHandler (1)

```
void startDocument()
void endDocument()
void startElement(String namespaceURI,
      String localName, String qName, Attributes atts)
void endElement(String namespaceURI, String localName,
      String qName)
void characters(char[] ch, int start, int length)
void setDocumentLocator(Locator locator)
void processingInstruction(String target, String data)
// target: z.B. "xml-stylesheet". data: unstrukturierter Rest
// d.h.: Pseudo-Attribute werden nicht erkannt wie z.B.:
// <?xml-stylesheet type="text/css" href="order.css"?>
```

Überblick: Methoden des ContentHandler (2)

```
void ignorableWhitespace(char[] ch, int start,
     int length)
// bei validierendem Parsers: meldet ignorable whitespace mit
// diesem Event und nicht als "characters" (meist reicht DTD)
void skippedEntity(String name)
// falls Parser die (externe) DTD nicht liest, meldet er
// eine Entity Referenz mit diesem Event (anstatt die
// Entity zu expandieren und als "characters" zu melden.
void startPrefixMapping(String prefix, String uri)
void endPrefixMapping(String prefix)
// nur interessant, wenn man auf ein Prefix in einem
// Attribut-Wert zugreift (z.B. bei XML-Schema)
```

Dokument-Verarbeitung

```
void startDocument()
void endDocument()
```

- Ein XMLReader + ContentHandler kann für die Verarbeitung mehrerer Dokumente (hintereinander!) verwendet werden.

  => Initialisierungen am besten in **`startDocument`**
- Wohlgeformtheitsfehler:
  - werden vom Parser erst erkannt, nachdem er schon etliche Events geliefert hat. => dessen muss man sich beim Verarbeiten von Events bewusst sein (d.h.: ev. Rückrollen erforderlich).
  - Allfälliger clean-up code in **`endDocument`** wird ev. nie ausgeführt.
- Reader ist weder thread-safe noch reantrant.

  => Parallele Verarbeitung von mehreren Dokumenten erfordert mehrere Reader Objekte.

Element-Verarbeitung

```
void startElement(String namespaceURI,
     String localName, String qName, Attributes atts)
void endElement(String namespaceURI, String localName,
     String qName)
```

- Argumente:
  - namespaceURI: leerer String bei Element ohne Namespace
  - qName = Präfix + ":" + localName
  - atts: siehe nächste Folie
- SAX hat absolut kein "Gedächtnis".

  => Häufig verwendete Datenstruktur in der Applikation:
  Stack für die offenen Elemente:
  - bei **`startElement`**: push Element-Informationen
  - bei **`endElement`**: pop

Attributes Interface

- `startElement` liefert Attribute als Attributes Objekt zurück.
- Zugriff mittels `getLength()` und Attribut-Index:

```
String getLocalName(int index)
String getURI(int index)
String getQName(int index)
String getType(int index)
```

- Zugriff mittels Namespace-qualified name:

```
int getIndex(String uri, String localName)
String getValue(String uri, String localName)
String getType(String uri, String localName)
```

- Analog: Zugriff mittels qualified (prefixed) name, z.B.:

```
int getIndex(String qualifiedName)
```

Text-Verarbeitung

```
void characters(char[] ch, int start, int length)
```

- Darstellung von Text-Inhalt:
  - als Char-Array mit Start-Index und Längenangabe
  - (insbes. bei langem Text): SAX-Parser darf Text auf beliebig viele hintereinander folgende **`characters`** Events aufteilen.
- Häufige Verarbeitungsart:
  - Text-Inhalt in einem StringBuffer akkumulieren
  - bei Element-Event: Ausgabe und Initialisierung eines StringBuffer

Locator Interface

- Das Locator Interface erlaubt Zugriff auf die Stelle im Dokument, wo das letzte Event gefunden wurde.
- SAX-Parser sollte (muss aber nicht) Locator implementieren.
- Methoden, z.B.:

```
int getLineNumber()
int getColumnNumber()
```

- Typischer Code, um Locator-Information verfügbar zu haben:
  - Instanz-Variable für ContentHandler Objekt definieren:

```
private Locator locator;
```

  - Referenz auf Locator abspeichern:

```
public void setDocumentLocator(Locator locator) {
   this.locator = locator;
 }
```

DefaultHandler

- Deklaration des DefaultHandlers (in org.xml.sax.helpers):

```
public class DefaultHandler extends Object
implements EntityResolver, DTDHandler,
    ContentHandler, ErrorHandler
```

- Enthält default-Deklarationen für alle callback Funktionen dieser 4 Event Handler
- Default-Verhalten: "do nothing"-Methoden
- Bequeme Definition eines eigenen Handlers:
  - von DefaultHandler erben
  - die tatsächlich benötigten callback Funktionen überschreiben

Weitere Event Handler

- EntityResolver
- DTDHandler
- ErrorHandler

EntityResolver

- Einzige Methode:

```
InputSource resolveEntity(String publicId, String systemId)
```

- Idee:
  - Die **`resolveEntity`**-Methode wird vom Parser aufgerufen, wenn eine externe geparste Entity gefunden wurde.
  - externe geparste Entities können mittels SystemId oder PublicId angegeben werden.
  - Mittels **`resolveEntity`** Methode bekommt die Applikation die Möglichkeit (insbes. bei PublicId) dem Parser eine andere InputSource bereitzustellen.

DTDHandler

- Methoden:

```
void notationDecl(String name, String publicId,
 String systemId)
void unparsedEntityDecl(String name, String publicId,
 String systemId, String notationName)
```

- Idee:
  - Während der Bearbeitung der DTD meldet der Parser die Deklarationen von Notations und unparsed Entities.
  - Die Applikation speichert sich diese Informationen in eigenen Datenstrukturen (z.B. in Hash Tabelle)
  - Wenn der Parser Attribute vom Typ "NOTATION", "ENTITY" oder "ENTITIES" meldet, hat die Applikation die nötigen Informationen.

ErrorHandler

- Methoden:

```
void fatalError(SAXParseException exception)
// non-recoverable error
void error(SAXParseException exception)
void warning(SAXParseException exception)
```

- Idee:
  - Bei Wohlgeformtheitsfehler wirft der Parser eine Exception und beendet den Parse-Vorgang.
  - Bei anderen Fehlern (insbes. Gültigkeitsfehler bei validierendem Parser) kann der Parser fortsetzen und wirft keine Exception.
  - Benachrichtigung der Applikation: Parser reicht SAXParseException an die entsprechende Methode des ErrorHandlers.

XMLReader

- Reader-Implementierung
- Features und Properties
- SAX-Ausgabe

XML Reader

- Ist der eigentliche SAX-Parser, d.h.: liest das XML-Dokument und ruft die callback Funktionen auf.
- Erlaubt das Setzen/Auslesen bestimmter Properties/Features:

  `setFeature, setProperty, getFeature, getProperty`
- Benutzer registriert die Event Handler (mit den callback Funktionen):

  `setContentHandler, setDTDHandler,`

  `setEntityResolver, setErrorHandler`
- Analog dazu get-Methoden für die Event Handler:

  `getContentHandler, getDTDHandler,` etc.
- Methode zum Anstoßen des Parsers:

  `parse`

Reader-Implementierung

- XMLReader Instanz mittels XMLReader Factory erzeugen
- Auswahl einer bestimmten Implementierung:
  - Default-Implementierung auswählen:

```
public static XMLReader createXMLReader()
    throws SAXException;
// Auswahl des SAX-Parsers laut system property
// org.xml.sax.driver (kann mit Kommandozeilen-
// parameter "-D" gesetzt werden)
```

  - Auswahl einer bestimmten Implementierung:

```
public static XMLReader createXMLReader(
    String className) throws SAXException;
// className = gewünschte Implementierung,
// z.B.: "org.apache.xerces.parsers.SAXParser"
```

Features und Properties

XMLReader-Methoden:

- `boolean getFeature(String name) throws SAXNotRecognizedException, SAXNotSupportedException`
- `Object getProperty(String name) throws SAXNotRecognizedException, SAXNotSupportedException`
- `void setFeature(String name, boolean value) throws SAXNotRecognizedException, SAXNotSupportedException`
- `void setProperty(String name, Object value) throws SAXNotRecognizedException, SAXNotSupportedException`

Features

- Haben einen boolean Wert: true / false
- Feature-Namen sind absolute URLs
- Standard-Features (z.B. validierend, Namespace-aware, …)

```
parser.setFeature(
 "http://xml.org/sax/features/validation", true);
parser.setFeature(
 "http://xml.org/sax/features/namespaces", true);
```

- Vendor-spezifische Features, z.B.:

```
boolean schemaValidierend = parser.getFeature(
 "http://apache.org/xml/features/validation/schema");
boolean schemaFullChecking = parser.setFeature(
 "http://apache.org/xml/features/validation/schema-full-
 checking");
```

Properties

- Haben einen "beliebigen" Typ, z.B.:
  - Property **`"http://xml.org/sax/properties/lexical-handler"`**
    (ermöglicht Zugriff auf Kommentare, CDATA Sections, …)
    => das konkrete handler-Objekt wird als Property gesetzt/gelesen.
- Property-Namen sind absolute URLs
- Weitere Standard-Properties, z.B.:
  - Property **`"http://xml.org/sax/properties/xml-string"`**
    read-only Property: liefert den Text-String, der das aktuelle SAX-Event ausgelöst hat.
- Vendor-spezifische Properties, z.B.:
  - Property **`"http://apache.org/xml/properties/schema/external-schemaLocation"`**: gibt an, wo der Parser nach XML-Schema Dateien suchen soll.

SAX-Filter

- Funktionsweise eines SAX-Filters
- Verwendung eines SAX-Filters

Funktionsweise eines SAX-Filters

- Filter beschränken/manipulieren auftretende Events
- Filter können verschachtelt verwendet werden
- Falls sich Dokumentstruktur ändert, kann eine Adaption des Filters ausreichend sein (also: ohne die Applikation zu ändern).
- Bemerkung: Filter kann auch für nicht XML-Input verwendet werden (ohne dass die Client-Applikation das merkt...)

Funktionsweise eines SAX-Filters (2)

- Vorbereitung:
  - Applikation teilt dem Filter mit, auf welchen Reader er horchen muss
  - Applikation registriert ihre Event Handler beim Filter
- Start des Parse-Vorgangs:
  - Applikation ruft parse()-Methode des Filters auf
  - Filter ruft parse()-Methode des Readers auf
- Parse-Vorgang:
  - Reader erzeugt Events => ruft callback Funktionen *des Filters* auf
  - Filter ruft innerhalb seiner callback Funktionen die callback Funktionen der Applikation auf.
- Der Filter ist also gleichzeitig Event Handler und Reader.

Verwendung eines SAX-Filters

- XMLFilter Interface: erweitert XMLReader um 2 Methoden:
  - `void setParent(XMLReader parent)` und `XMLReader getParent()`
  - "parent" = Reader, auf den der Filter horchen muss
  - Mittels **`setContentHandler`**, etc. werden die Event Handler der Applikation beim Filter registriert.
- Implementierung des XMLFilter Interface:
  - "per Hand": ziemlich aufwändig (XMLReader hat 14 Methoden)
  - Eleganterer Weg: mittels XSLT Stylesheet kann ein XMLFilter automatisch erzeugt werden (späterer VL-Termin).
  - Die Klasse XMLFilterImpl in org.xml.sax.helpers stellt einen Default-Filter bereit, der die Requests in beiden Richtungen transparent durchreicht.