

# Extensible Markup Language



X

M

L

# General Advantages of XML for KR

XML offers new general possibilities, from which AI knowledge representation (KR) can profit:

- (1) Definition of self-describing data in worldwide standardized, non-proprietary format
- (2) Structured data and knowledge exchange for enterprises in various industries
- (3) Integration of information from different sources (into uniform documents)



# Specific Advantages of XML for KR

XML provides the most suitable infrastructure for knowledge bases on the Web (incl. for W3C languages such as RDF)

Additional special KR uses of XML are:

- Uniform storage of knowledge bases
- Interchange of knowledge bases between different AI languages
- Exchange between knowledge bases and databases, application systems, etc.

Even transformation/compilation of AI source programs using XML markup and annotations is possible



# Address Example: External to HTML

## External Presentation:

*Xaver M. Linde*  
Wikingerufer 7  
**10555 Berlin**

*HTML tags are still  
presentation-oriented*

## HTML Markup:

```
<em>Xaver M. Linde</em>  
<br>  
Wikingerufer 7  
<br>  
<strong>10555 Berlin</strong>
```



# Address Example: HTML to XML

## HTML Markup:

```
<em>Xaver M. Linde</em>  
<br>  
Wikingerufer 7  
<br>  
<strong>10555 Berlin</strong>
```

While not conveying  
any formal semantics:

*XML tags are chosen for  
content-structuring needs*

## XML Markup:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```



# Address Example: XML to External

## XML Markup:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```

*XML stylesheets are, e.g., usable to generate different presentations*

## External Presentations:

*Xaver M. Linde*  
*Wikingerufer 7*  
**10555 Berlin**

...

**Xaver M. Linde**  
**Wikingerufer 7**  
10555 Berlin



# Address Example: XML to XML

## XML Markup 1:

```
<address>  
  <name>Xaver M. Linde</name>  
  <street>Wikingerufer 7</street>  
  <town>10555 Berlin</town>  
</address>
```

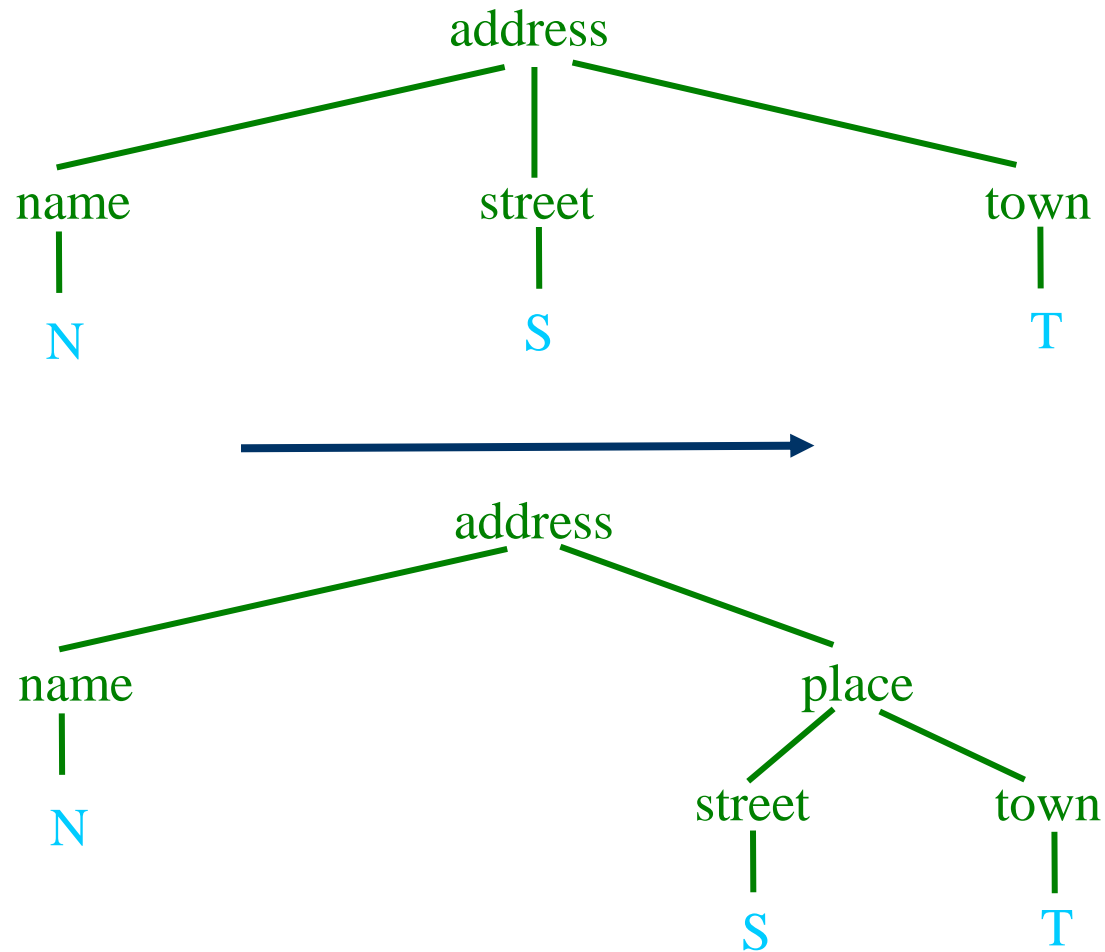
## XML Markup 2:

```
<address>  
  <name>Xaver M. Linde</name>  
  <place>  
    <street>Wikingerufer 7</street>  
    <town>10555 Berlin</town>  
  </place>  
</address>
```

*XML stylesheets are also usable to transform XML representations*



# Address Example: Some Stylesheets Will Contain Term-(Tree-)Rewriting Rules



X

M

L

# Address Example: XML Queries

## XML Markup:

*e*  
*l*  
*e*  
*m*  
*e*  
*n*  
*t*

```
<address>  subelements
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
</address>
```

## XML Query (XML-QL):

WHERE

```
<address>
  <name>Xaver M. Linde</name>
  <street>$s</street>
  <town>$t</town>
</address>
```

CONSTRUCT

```
<binding>
  <s>$s</s>
  <t>$t</t>
</binding>
```

*XML queries can  
select subelements  
of XML elements*

```
<binding>
  <s>Wikingerufer 7</s>
  <t>10555 Berlin</t>
</binding>
```

# Address Example: Prolog Queries

## Prolog Term:

*s*  
*t*  
*r*  
*u*  
*c*  
*t*  
*u*  
*r*  
*e*

```
address( substructures
  name("Xaver M. Linde"),
  street("Wikingerufer 7"),
  town("10555 Berlin")
)
```

## Prolog Query:

```
address(
  name("Xaver M. Linde"),
  street(S),
  town(T)
)
```

*Prolog queries can  
select substructures  
of Prolog structures*

S = "Wikingerufer 7"  
T = "10555 Berlin"

# Address Example: The Element Tree

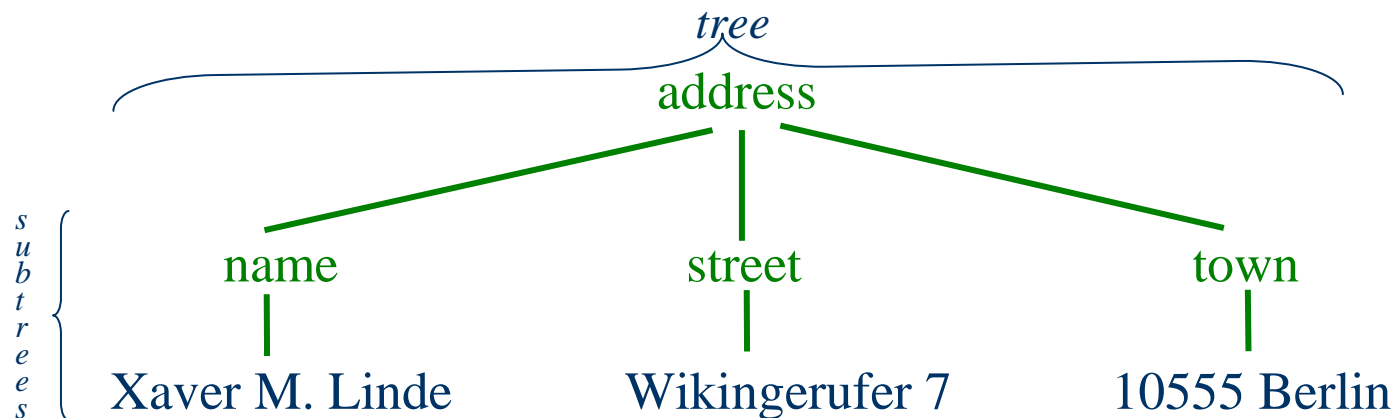
## XML Markup:

*e*  
*l*  
*e*  
*m*  
*e*  
*n*  
*t* { `<address>` *subelements*  
    `<name>Xaver M. Linde</name>`  
    `<street>Wikingerufer 7</street>`  
    `<town>10555 Berlin</town>`  
    `</address>`

## Prolog Term:

*s*  
*t*  
*r*  
*u*  
*c*  
*t*  
*u*  
*r*  
*e* { `address(` *substructures*  
    `name("Xaver M. Linde"),`  
    `street("Wikingerufer 7"),`  
    `town("10555 Berlin")`  
    `)`

## Node-Labeled, (Left-to-Right-)Ordered Element Tree:



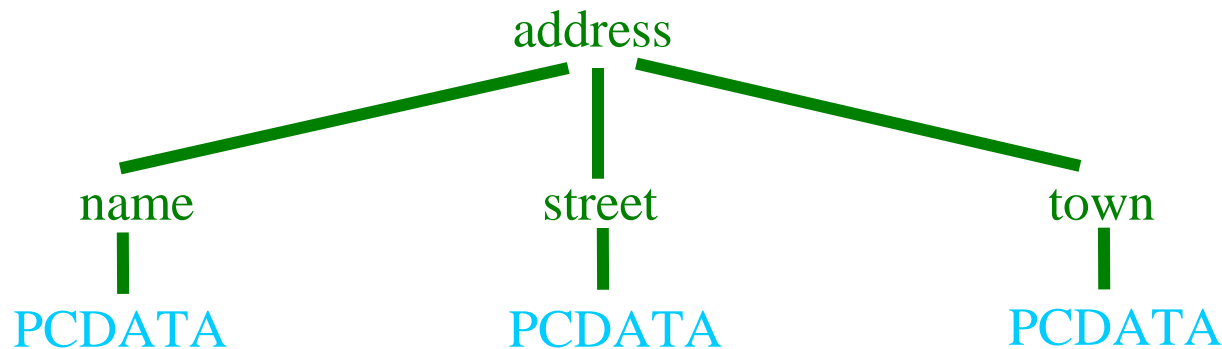
# Address Example: Document Type Definition and Tree (1)

Document Type Definition (DTD):

Extended Backus-Naur Form (EBNF):

<!ELEMENT address	(name, street, town) >	address ::= name street town
<!ELEMENT name	(#PCDATA) >	name ::= PCDATA
<!ELEMENT street	(#PCDATA) >	street ::= PCDATA
<!ELEMENT town	(#PCDATA) >	town ::= PCDATA

Document Type Tree:

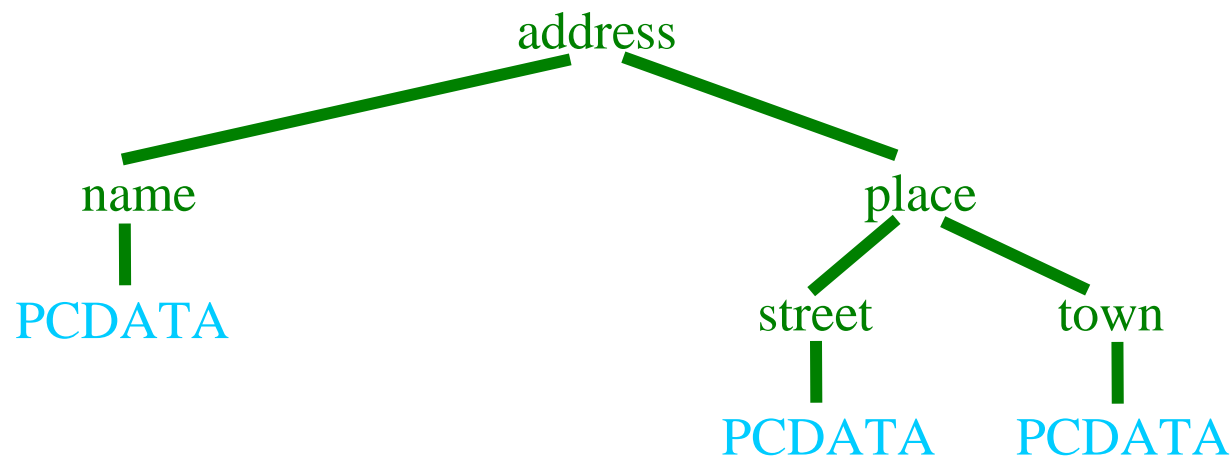


# Address Example: Document Type Definition and Tree (2)

## Document Type Definition (DTD):

```
<!ELEMENT address (name, place) >  
<!ELEMENT place (street, town) >  
<!ELEMENT name (#PCDATA) >  
<!ELEMENT street (#PCDATA) >  
<!ELEMENT town (#PCDATA) >
```

## Document Type Tree:



# Well-Formedness and Validity

XML principles for a document being *well-formed*:

- Open and close all tags
- Empty tags end with />
- There is a unique root element
- Elements may not overlap
- Attribute values are quoted
- < and & are only used to start tags and entities
- Only the five predefined entity references are used

XML principle for a document being *valid* with respect to (w.r.t.) a DTD :

- Match the constraints listed in the DTD (or, generate from DTD as linearized derivation tree, as shown later)

Checked by **validators** such as

<http://www.stg.brown.edu/service/xmlvalid/>



# Mail-Box Example: Address Variant

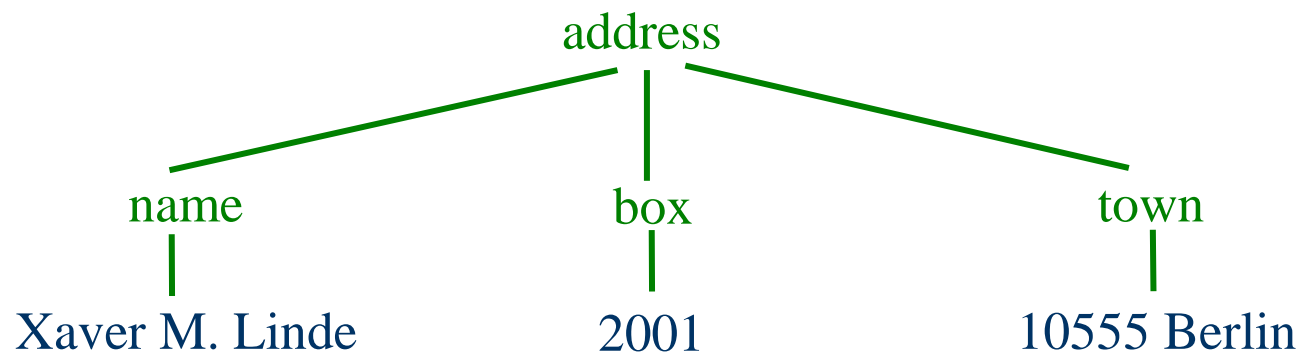
## XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <box>2001</box>
  <town>10555 Berlin</town>
</address>
```

## Prolog Term:

```
address(
  name("Xaver M. Linde"),
  box("2001"),
  town("10555 Berlin")
)
```

## Node-Labeled, (Left-to-Right-)Ordered Element Tree:



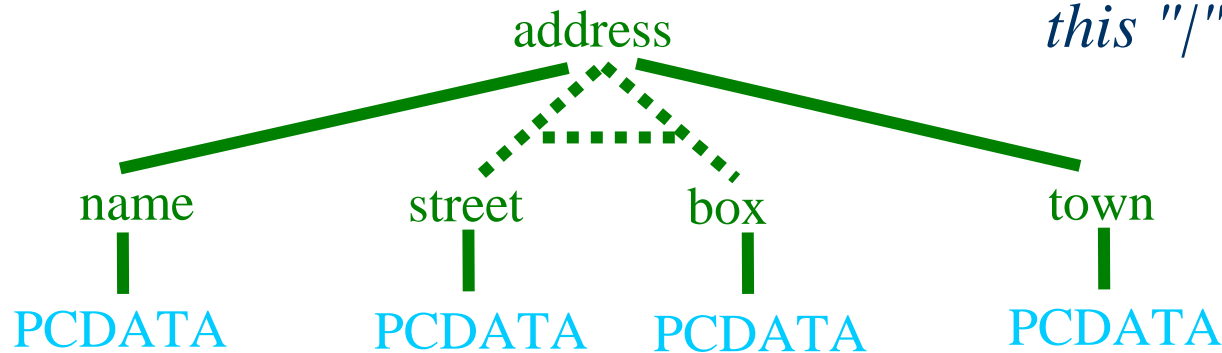
# "|" - Disjoined Street/Mail-Box Example: Document Type Definition and Tree

## Document Type Definition (DTD):

```
<!ELEMENT address      (name, (street | box), town) >  
<!ELEMENT name        (#PCDATA) >  
<!ELEMENT street      (#PCDATA) >  
<!ELEMENT box         (#PCDATA) >  
<!ELEMENT town        (#PCDATA) >
```

*"/": Choice*  
*The above box address and the original street address are valid w.r.t. this "/"-DTD*

## Document Type Tree:



# Phone & Fax Example: Address Variant

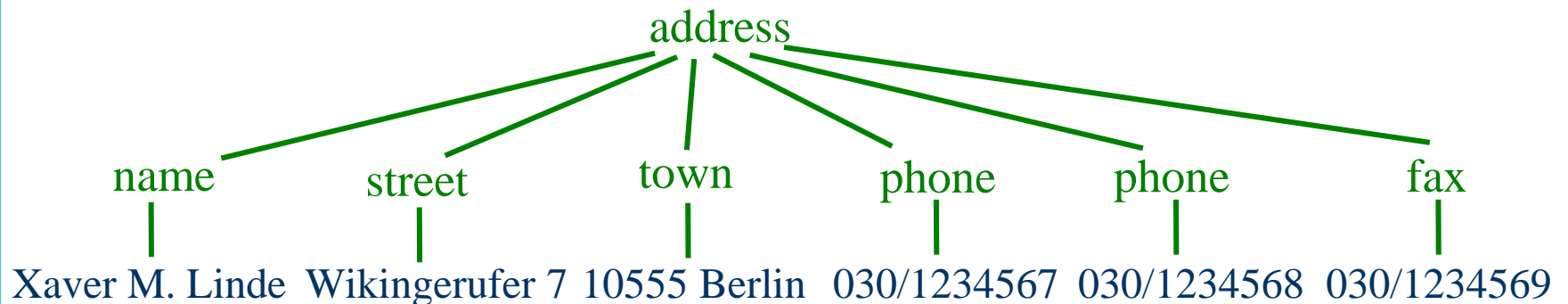
## XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <phone>030/1234567</phone>
  <phone>030/1234568</phone>
  <fax>030/1234569</fax>
</address>
```

## Prolog Term:

```
address(
  name("Xaver M. Linde"),
  street("Wikingerufer 7"),
  town("10555 Berlin"),
  phone("030/1234567"),
  phone("030/1234568"),
  fax("030/1234569")
)
```

## Node-Labeled, (Left-to-Right-)Ordered Element Tree:



# "+"/"\*" -Repetitive-Phone & -Fax Example: Document Type Definition and Tree

## Document Type Definition (DTD):

<!ELEMENT address (name, street, town, phone+, fax\*) >

<!ELEMENT name (#PCDATA) >

<!ELEMENT street (#PCDATA) >

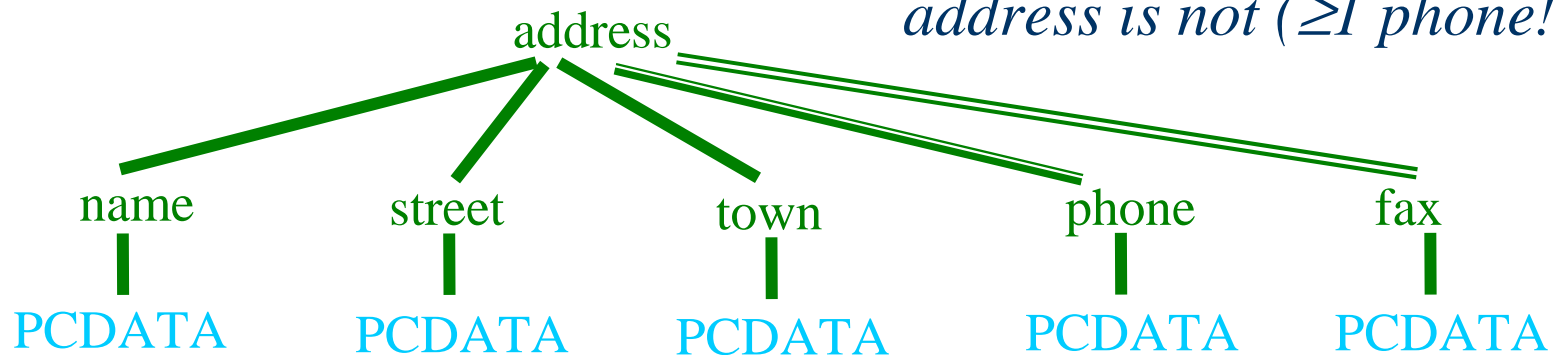
<!ELEMENT town (#PCDATA) >

<!ELEMENT phone (#PCDATA) >

<!ELEMENT fax (#PCDATA) >

*"+"/"\*" : One/Zero or More  
The above two-phone/one-fax  
address is valid w.r.t. this  
"+"/"\*" -DTD but the  
original no-phone/no-fax  
address is not ( $\geq 1$  phone!)*

## Document Type Tree:



# Country Example: Address Variant

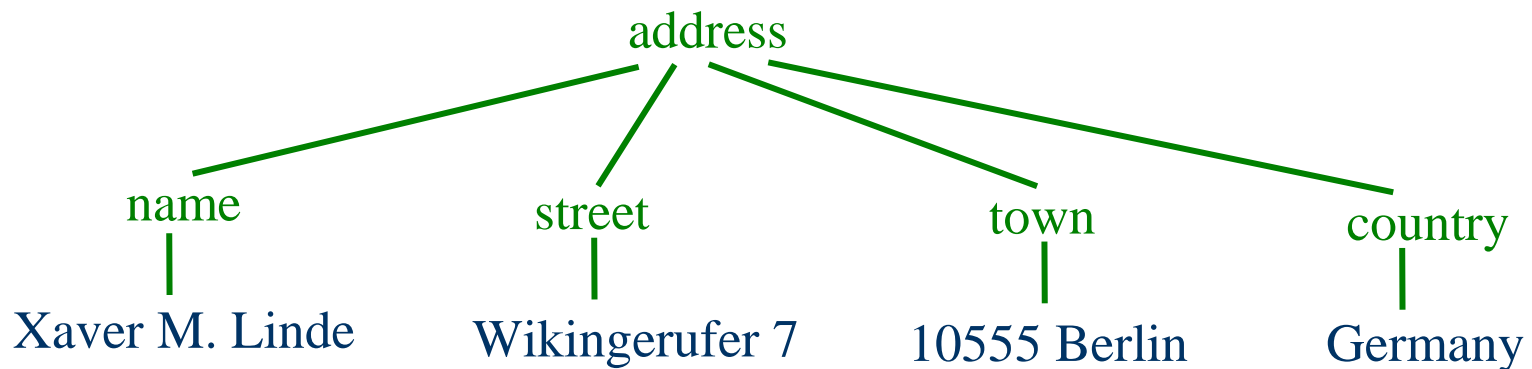
## XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <country>Germany</country>
</address>
```

## Prolog Term:

```
address(
  name("Xaver M. Linde"),
  street("Wikingerufer 7"),
  town("10555 Berlin"),
  country("Germany")
)
```

## Node-Labeled, (Left-to-Right-)Ordered Element Tree:



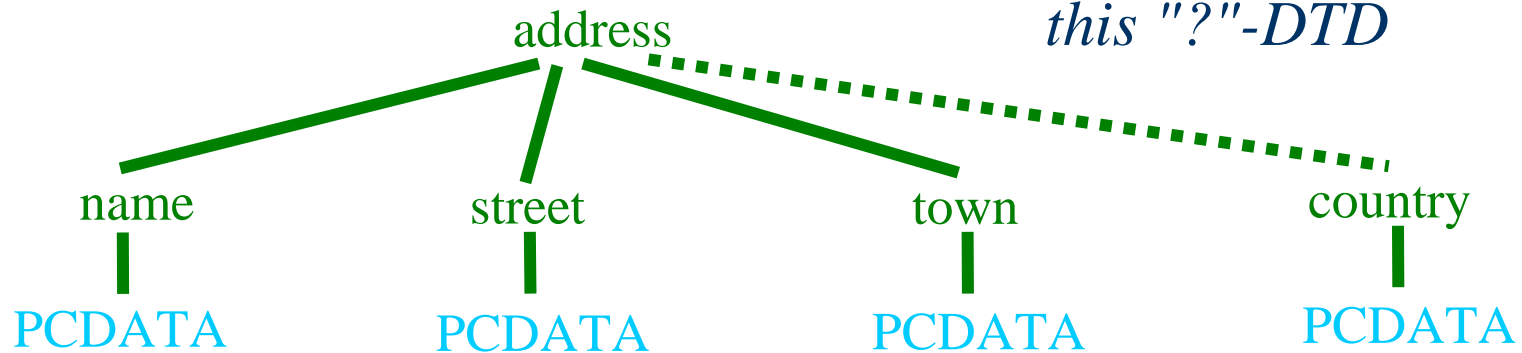
# "?"-Optional-Country Example: Document Type Definition and Tree

## Document Type Definition (DTD):

```
<!ELEMENT address (name, street, town, country?) >  
<!ELEMENT name (#PCDATA) >  
<!ELEMENT street (#PCDATA) >  
<!ELEMENT town (#PCDATA) >  
<!ELEMENT country (#PCDATA) >
```

*"?": One or Zero  
The above country  
address and the  
original countriless  
address are valid w.r.t.  
this "?"-DTD*

## Document Type Tree:



# Country Address: A Complete XML Document Referring to an External DTD

XML Document (just ASCII, e.g. stored in a file):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE address SYSTEM "country-address.dtd">
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <country>Germany</country>
</address>
```

The *XML declaration* uses standalone attribute with "no" value: DTD import

The *DOCUMENT TYPE declaration* names the *root element* address and, after the *SYSTEM keyword*, refers to an *external DTD* "country-address.dtd"

(or, at some absolute URL, to an "http://www.test.org/country-address.dtd")



# "minOccurs"-Optional-Country Example: XML Schema Definition

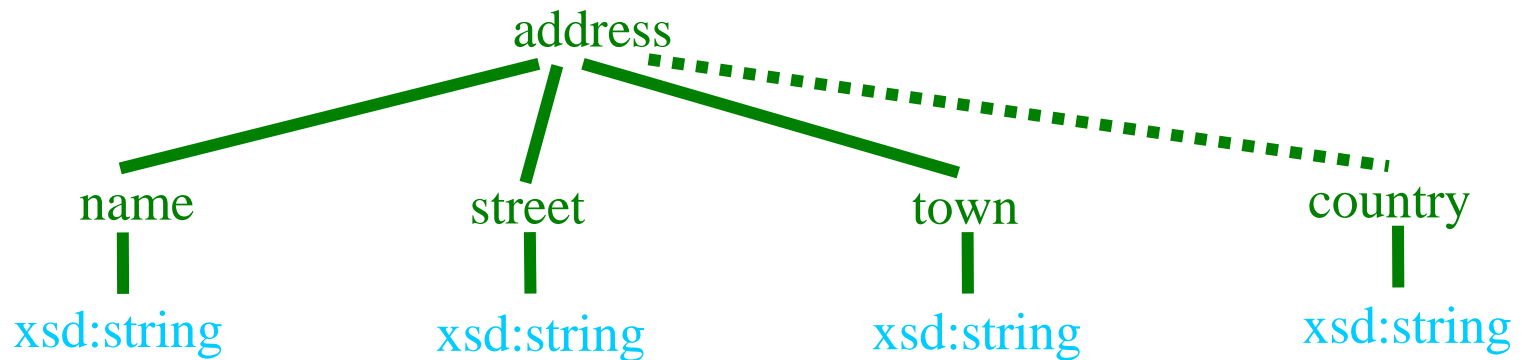
## Equivalent XML Schema Definition (XSD):

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="address">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="town" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



# "minOccurs"-Optional-Country Example: XML Schema Tree

Equivalent Document Schema Tree:



# Country Address: A Complete XML Document Referring to an External XSD

Equivalent XML Document (just ASCII, e.g. stored in a file):

```
<?xml version="1.0"?>
<address
  xsi:noNamespaceSchemaLocation="country-address.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <country>Germany</country>
</address>
```

The *xsi:noNamespaceSchemaLocation* attribute is embedded in the *root element* address and refers to an *external XSD* "country-address.xsd" (or, at some absolute URL, to an "http://www.test.org/country-address.xsd")

