

XML

Document Type Definitions
XML Schema

Well-Formed and Valid XML

- ◆ *Well-Formed XML* allows you to invent your own tags.
- ◆ *Valid XML* conforms to a certain DTD.

Well-Formed XML

- ◆ Start the document with a *declaration*, surrounded by `<?xml ... ?>` .

- ◆ Normal declaration is:

```
<?xml version = "1.0"  
standalone = "yes" ?>
```

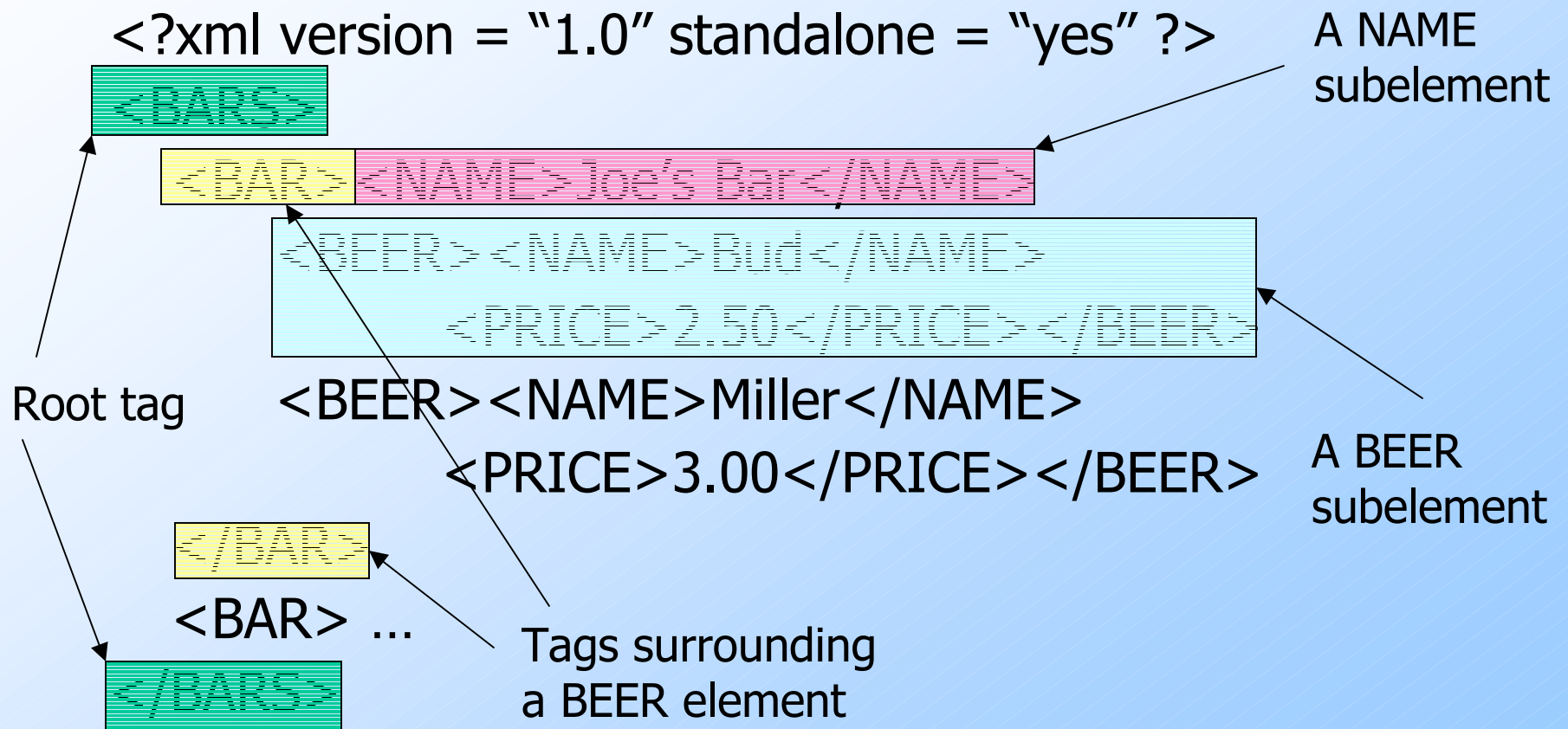
- ◆ "standalone" = "no DTD provided."

- ◆ Balance of document is a *root tag* surrounding nested tags.

Tags

- ◆ Tags are normally matched pairs, as `<FOO> ... </FOO>`.
- ◆ Unmatched tags also allowed, as `<FOO/>`
- ◆ Tags may be nested arbitrarily.
- ◆ XML tags are case-sensitive.

Example: Well-Formed XML



DTD Structure

```
<!DOCTYPE <root tag> [  
  <!ELEMENT <name> (<components>) >  
  . . . more elements . . .  
>
```

DTD Elements

- ◆ The description of an element consists of its name (tag), and a parenthesized description of any nested tags.
 - ◆ Includes order of subtags and their multiplicity.
- ◆ Leaves (text elements) have #PCDATA (*Parsed Character DATA*) in place of nested tags.

Example: DTD

```
<!DOCTYPE BARS [
```

```
<!ELEMENT BARS (BAR*)>
```

A BARS object has zero or more BAR's nested within.

```
<!ELEMENT BAR (NAME, BEER+)>
```

A BAR has one NAME and one or more BEER subobjects.

```
<!ELEMENT NAME (#PCDATA)>
```

```
<!ELEMENT BEER (NAME, PRICE)>
```

A BEER has a NAME and a PRICE.

```
<!ELEMENT PRICE (#PCDATA)>
```

```
]>
```

NAME and PRICE are text.

Element Descriptions

- ◆ Subtags must appear in order shown.
- ◆ A tag may be followed by a symbol to indicate its multiplicity.
 - ◆ * = zero or more.
 - ◆ + = one or more.
 - ◆ ? = zero or one.
- ◆ Symbol | can connect alternative sequences of tags.

Example: Element Description

- ◆ A name is an optional title (e.g., "Prof."), a first name, and a last name, in that order, or it is an IP address:

```
<!ELEMENT NAME (  
    (TITLE?, FIRST, LAST) | IPADDR  
)>
```

Use of DTD's

1. Set standalone = "no".
2. Either:
 - a) Include the DTD as a preamble of the XML document, or
 - b) Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.

Example: (a)

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE BARS [  
  <ELEMENT BARS (BAR*)>  
  <ELEMENT BAR (NAME, BEER+)>  
  <ELEMENT NAME (#PCDATA)>  
  <ELEMENT BEER (NAME, PRICE)>  
  <ELEMENT PRICE (#PCDATA)>  
>
```

The DTD

The document

```
<BARS>  
  <BAR> <NAME>Joe's Bar</NAME>  
    <BEER> <NAME>Bud</NAME> <PRICE>2.50</PRICE> </BEER>  
    <BEER> <NAME>Miller</NAME> <PRICE>3.00</PRICE> </BEER>  
  </BAR>  
  <BAR> ...  
</BARS>
```

Example: (b)

◆ Assume the BARS DTD is in file bar.dtd.

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE BARS SYSTEM "bar.dtd">
```

```
<BARS>
```

```
  <BAR><NAME>Joe's Bar</NAME>
```

```
    <BEER><NAME>Bud</NAME>
```

```
      <PRICE>2.50</PRICE></BEER>
```

```
    <BEER><NAME>Miller</NAME>
```


```
      <PRICE>3.00</PRICE></BEER>
```

```
  </BAR>
```

```
  <BAR> ...
```

```
</BARS>
```

Get the DTD
from the file
bar.dtd



Attributes

◆ Opening tags in XML can have *attributes*.

◆ In a DTD,

```
<!ATTLIST E . . . >
```

declares attributes for element *E*, along with its datatype.

Example: Attributes

- ◆ Bars can have an attribute `kind`, a character string describing the bar.

```
<!ELEMENT BAR (NAME BEER*) >
```

```
<!ATTLIST BAR kind
```

```
#IMPLIED >
```

```
CDATA
```

Attribute is optional
opposite: #REQUIRED

Character string
type; no tags

Example: Attribute Use

- ◆ In a document that allows BAR tags, we might see:

```
<BAR kind = "sushi">  
  <NAME>Homma 's</NAME>  
  <BEER><NAME>Sapporo</NAME>  
    <PRICE>5.00</PRICE></BEER>  
  ...  
</BAR>
```


ID's and IDREF's

- ◆ Attributes can be pointers from one object to another.
 - ◆ Compare to HTML's NAME = "foo" and HREF = "#foo".
- ◆ Allows the structure of an XML document to be a general graph, rather than just a tree.

Creating ID's

- ◆ Give an element E an attribute A of type ID.
- ◆ When using tag $\langle E \rangle$ in an XML document, give its attribute A a unique value.
- ◆ Example:

$\langle E \quad A = \text{"xyz"} \rangle$

Creating IDREF's

- ◆ To allow elements of type F to refer to another element with an ID attribute, give F an attribute of type IDREF.
- ◆ Or, let the attribute have type IDREFS, so the F -element can refer to any number of other elements.

Example: ID's and IDREF's

- ◆ A new BARS DTD includes both BAR and BEER subelements.
- ◆ BARS and BEERS have ID attributes `name`.
- ◆ BARS have SELLS subelements, consisting of a number (the price of one beer) and an IDREF `theBeer` leading to that beer.
- ◆ BEERS have attribute `soldBy`, which is an IDREFS leading to all the bars that sell it.

The DTD

Bar elements have name as an ID attribute and have one or more SELLS subelements.

```
<!DOCTYPE BARS [  
  <!ELEMENT BARS (BAR*, BEER*)>  
  <!ELEMENT BAR (SELLS+)>  
    <!ATTLIST BAR name ID #REQUIRED>  
  <!ELEMENT SELLS (#PCDATA)>  
    <!ATTLIST SELLS theBeer IDREF #REQUIRED>  
  <!ELEMENT BEER EMPTY>  
    <!ATTLIST BEER name ID #REQUIRED>  
    <!ATTLIST BEER soldBy IDREFS #IMPLIED>  
>
```

SELLS elements have a number (the price) and one reference to a beer.

Explained next

Beer elements have an ID attribute called name, and a soldBy attribute that is a set of Bar names.

Example: A Document

<BARS>

<BAR name = "JoesBar">

<SELLS theBeer = "Bud">2.50</SELLS>

<SELLS theBeer = "Miller">3.00</SELLS>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar
SuesBar ..." /> ...

</BARS>

Empty Elements

- ◆ We can do all the work of an element in its attributes.
 - ◆ Like BEER in previous example.
- ◆ **Another example:** SELLS elements could have attribute `price` rather than a value that is a price.

Example: Empty Element

◆ In the DTD, declare:

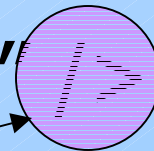
```
<!ELEMENT SELLS EMPTY>
```

```
<!ATTLIST SELLS theBeer IDREF #REQUIRED>
```

```
<!ATTLIST SELLS price CDATA #REQUIRED>
```

◆ Example use:

```
<SELLS theBeer = "Bud" price = "2.50">
```



Note exception to
"matching tags" rule

XML Schema

- ◆ A more powerful way to describe the structure of XML documents.
- ◆ XML-Schema declarations are themselves XML documents.
 - ◆ They describe “elements” and the things doing the describing are also “elements.”

Structure of an XML-Schema Document

```
<? xml version = ... ?>
```

```
<xschema xmlns:xschema =  
  "http://www.w3.org/2001/XMLSchema">  
  . . .
```

```
</xschema>
```

So uses of "xs" within the schema element refer to tags from this namespace.

Defines "xs" to be the *namespace* described in the URL shown. Any string in place of "xs" is OK.

The `xs:element` Element

- ◆ Has attributes:
 1. `name` = the tag-name of the element being defined.
 2. `type` = the type of the element.
 - ◆ Could be an XML-Schema type, e.g., `xs:string`.
 - ◆ Or the name of a type defined in the document itself.

Example: xs:element

```
<xs:element name = "NAME"  
  type = "xs:string" />
```

- ◆ Describes elements such as
`<NAME>Joe's Bar</NAME>`

Complex Types

- ◆ To describe elements that consist of subelements, we use `xs:complexType`.
 - ◆ Attribute `name` gives a name to the type.
- ◆ Typical subelement of a complex type is `xs:sequence`, which itself has a sequence of `xs:element` subelements.
 - ◆ Use `minOccurs` and `maxOccurs` attributes to control the number of occurrences of an `xs:element`.

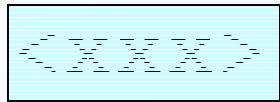
Example: a Type for Beers

```
<xs:complexType name = "beerType">  
  <xs:sequence>  
    <xs:element name = "NAME"  
      type = "xs:string"  
      minOccurs = "1" maxOccurs = "1" />  
    <xs:element name = "PRICE"  
      type = "xs:float"  
      minOccurs = "0" maxOccurs = "1" />  
  </xs:sequence>  
</xs:complexType>
```

Exactly one occurrence

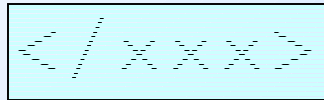
Like ? in a DTD

An Element of Type beerType



<NAME>Bud</NAME>

<PRICE>2.50</PRICE>



We don't know the
name of the element
of this type.

Example: a Type for Bars

```
<xs:complexType name = "barType">
  <xs:sequence>
    <xs:element name = "NAME"
      type = "xs:string"
      minOccurs = "1" maxOccurs = "1" />
    <xs:element name = "BEER"
      type = "beerType"
      minOccurs = "0" maxOccurs =
        "unbounded" />
  </xs:sequence>
</xs:complexType>
```

Like * in a DTD

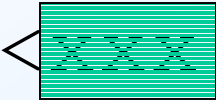
xs:attribute

- ◆ **xs:attribute** elements can be used within a complex type to indicate attributes of elements of that type.
- ◆ attributes of **xs:attribute**:
 - ◆ **name** and **type** as for **xs.element**.
 - ◆ **use** = "required" or "optional".

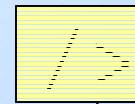
Example: xs:attribute

```
<xs:complexType name = "beerType">  
  <xs:attribute name = "name"  
    type = "xs:string"  
    use = "required" />  
  <xs:attribute name = "price"  
    type = "xs:float"  
    use = "optional" />  
</xs:complexType>
```

An Element of This New Type beerType

 name = "Bud"
price = "2.50"

We still don't know the
element name.



The element is
empty, since there
are no declared
subelements.

Restricted Simple Types

- ◆ `xs:simpleType` can describe enumerations and range-restricted base types.
- ◆ `name` is an attribute
- ◆ `xs:restriction` is a subelement.

Restrictions

- ◆ Attribute **base** gives the simple type to be restricted, e.g., `xs:integer`.
- ◆ `xs:{min, max}{Inclusive, Exclusive}` are four attributes that can give a lower or upper bound on a numerical range.
- ◆ `xs:enumeration` is a subelement with attribute **value** that allows enumerated types.

Example: **license** Attribute for BAR

```
<xs:simpleType name = "license">  
  <xs:restriction base = "xs:string">  
    <xs:enumeration value = "Full" />  
    <xs:enumeration value = "Beer only" />  
    <xs:enumeration value = "Sushi" />  
  </xs:restriction>  
</xs:simpleType>
```

Example: Prices in Range [1,5)

```
<xs:simpleType name = "price">  
  <xs:restriction  
    base = "xs:float"  
    minInclusive = "1.00"  
    maxExclusive = "5.00" />  
</xs:simpleType>
```

Keys in XML Schema

- ◆ An `xs:element` can have an `xs:key` subelement.
- ◆ **Meaning**: within this element, all subelements reached by a certain *selector* path will have unique values for a certain combination of *fields*.
- ◆ **Example**: within one BAR element, the `name` attribute of a BEER element is unique.

Example: Key

```
<xs:element name = "BAR" ... >
    . . .
    <xs:key name = "barKey">
        <xs:selector xpath = "BEER" />
        <xs:field xpath = "@name" />
    </xs:key>
    . . .
</xs:element>
```

And @ indicates an attribute rather than a tag.

XPath is a query language for XML. All we need to know here is that a path is a sequence of tags separated by /.

Foreign Keys

- ◆ An `xs:keyref` subelement within an `xs:element` says that within this element, certain values (defined by selector and field(s), as for keys) must appear as values of a certain key.

Example: Foreign Key

- ◆ Suppose that we have declared that subelement NAME of BAR is a key for BARS.
 - ◆ The name of the key is barKey.
- ◆ We wish to declare DRINKER elements that have FREQ subelements. An attribute **bar** of FREQ is a foreign key, referring to the NAME of a BAR.

Example: Foreign Key in XML Schema

```
<xs:element name = "DRINKERS"  
  . . .  
  <xs:keyref name = "barRef"  
    refers = "barKey"  
    <xs:selector xpath =  
      "DRINKER/FREQ" />  
    <xs:field xpath = "@bar" />  
  </xs:keyref>  
</xs:element>
```