Testing XML technologies

Software Testing and Quality Assurance

Xml example

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonList Type="Employee">
<Title> Value="Employee List"></Title>
<Contents>
<Employee>
<Name>John Barrimore</Name>
<No>18316</No>
<Deptno>d1</Deptno>
<Address>
<City>Seattle</City>
<Street>Abbey Rd</Street>
</Address>
</Employee>
</Contents>
</PersonList>
```

DTD

Document Type Definition

An optional prolog.

Xml Declaration

Processing Instructions

A Document Type Declaration

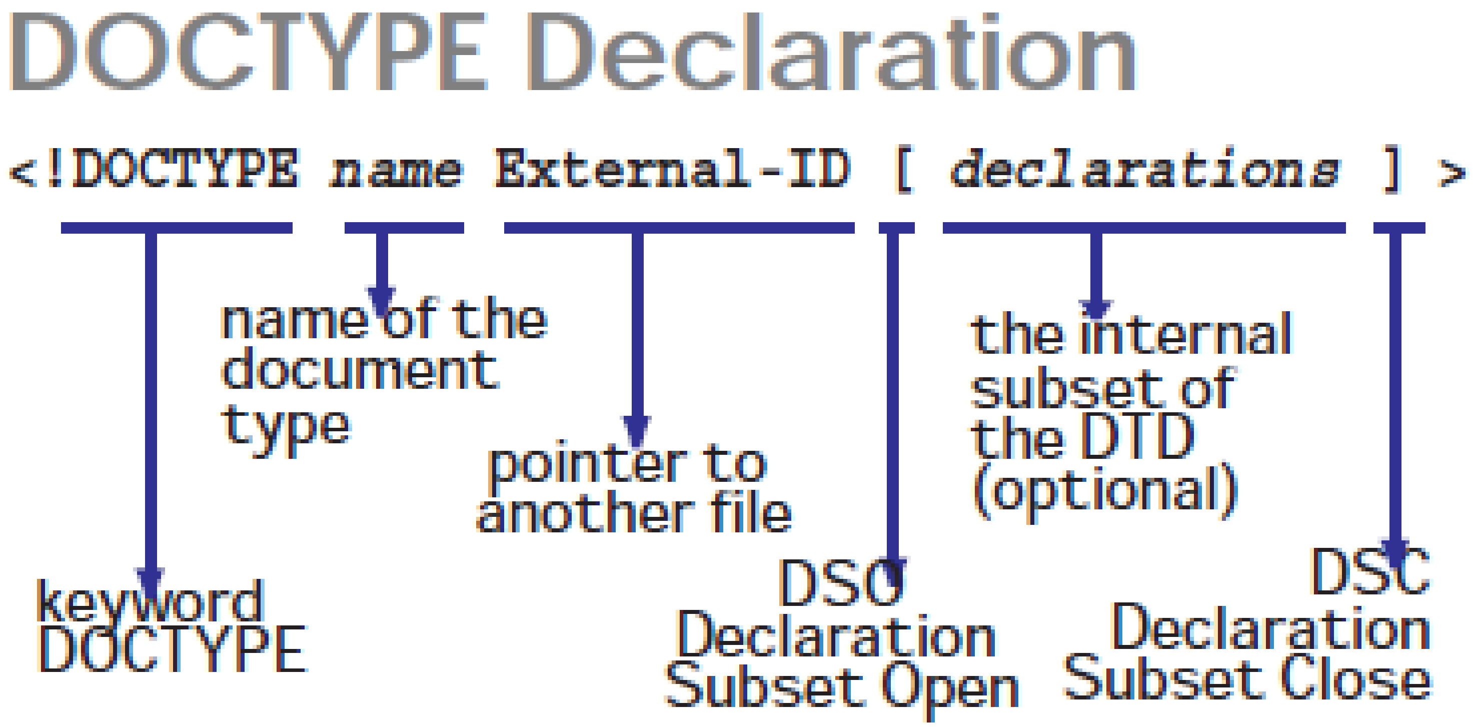
Processing Instructions

- <?xml-stylesheet type="text/css“ href="file.css"?>

A Document Type Declaration (DTD)

- The purpose of a Document Type Definition or DTD is to define the **structure of a document** encoded in XML (extended Markup Language).
  - It’s file that **constrains** or restricts certain elements and attributes to exist in XML document.

DTD


DOCTYPE Declaration
<!DOCTYPE name External-ID [declarations] >
keyword DOCTYPE
name of the document type
pointer to another file
DSO Declaration Subset Open
the internal subset of the DTD (optional)
DSC Declaration Subset Close

Well Formed XML

- **Well Formed** - The logical structure is not validated against the DTD. A well formed document follows a set of rules to qualify as "well formed".

Summary

- There must be one and only one document element.
- Every open tag must be closed.
- If an element is empty, it still must be closed.
  - Poorly-formed: <tag>
  - Well-formed: <tag></tag>
  - Also well-formed: <tag />

Summary

- Elements must be properly nested.
  - Poorly-formed: <a><b></a></b>
  - Well-formed: <a><b></b></a>
- Tag and attribute names are case sensitive.
- Attribute values must be enclosed in single or double quotes.

Valid xml

- A valid XML document is not the same as a well formed XML document.
- The first rule, for a valid XML document, is that it must be well formed .

Valid xml

- The second rule is that a valid XML document must conform to a document type.
- Rules that defines legal elements and attributes for XML documents are often called Document Type Definition (DTD) OR XML Schema

Is a Wellformed Document Valid?

- An example of a document that is well-formed but not valid based upon the XML grammar.

```
<paragraph>
  <p>Example of Well-formed HTML</p>
  <head>
    <title>Example</title>
  </head>
  <zorko>What is this?</zorko>
</paragraph>
```

DTD example

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Xml with dtd

-
  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE note SYSTEM "Note.dtd">
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
  </note>
  ```

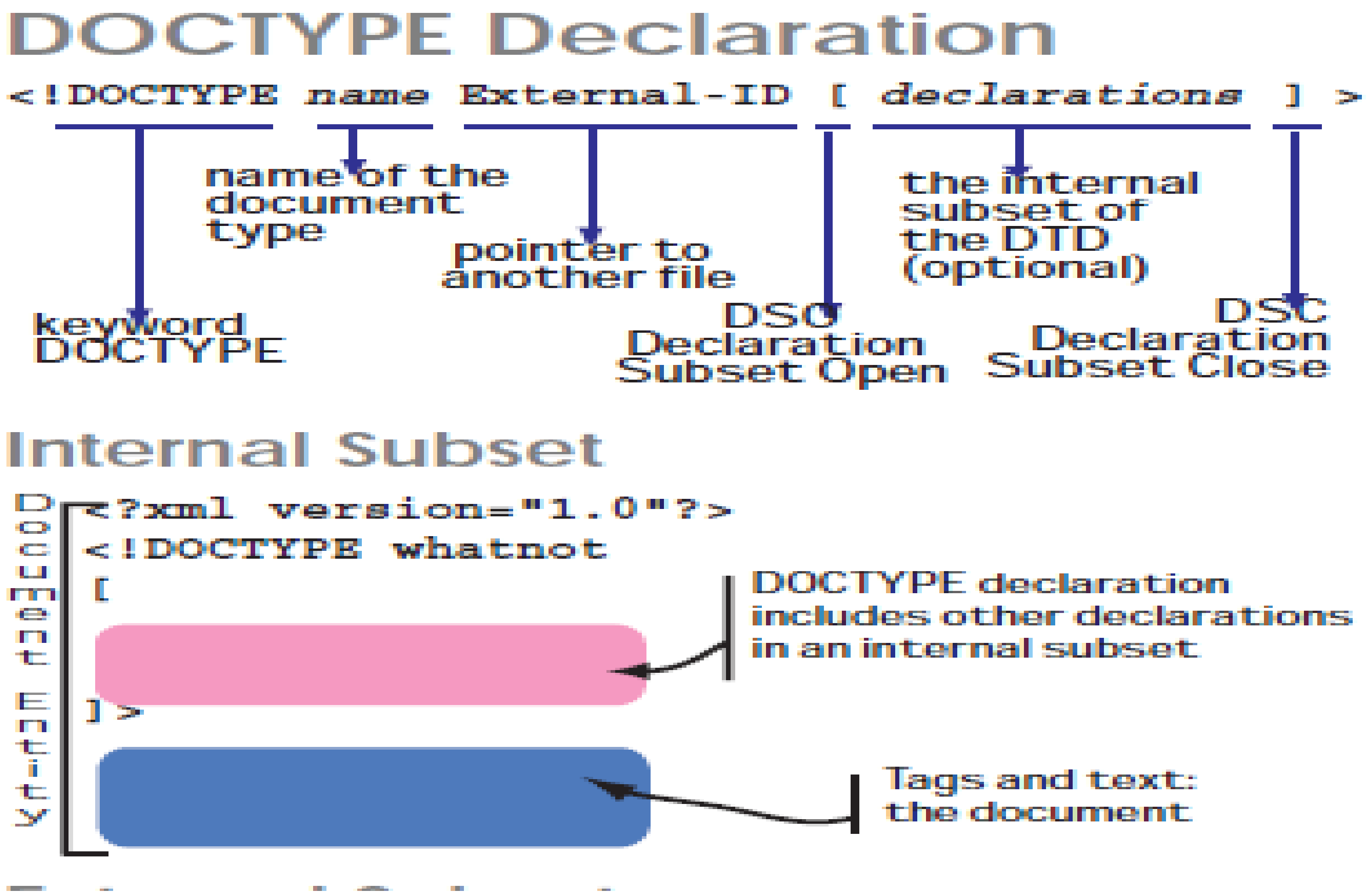
Why DTD ?

- **It defines the document structure with a list of legal elements and attributes.**
- your XML files can carry a description of its own format.
- independent groups of people can agree on a standard for interchanging data.
- you can verify that the data you receive from the outside world is valid.

DTD Declaration

- In general we can say that there are two main types of DTDs:
  - Internal
  - External
- Internal DTDs reside within the XML instance file
- External DTDs reside in an separate DTD document.
- Later we will find out how to merge the two to formulate a mixed DTD.

Internal DTD Declaration

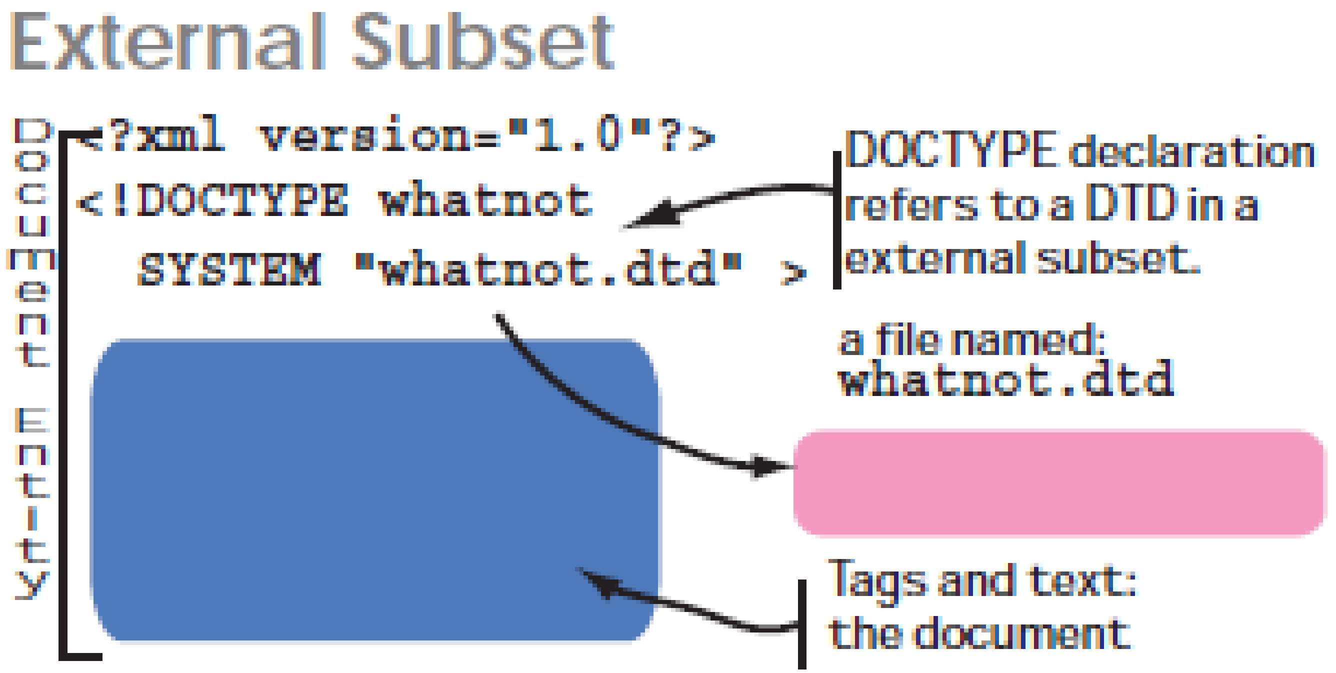

DOCTYPE Declaration
<!DOCTYPE name External-ID [declarations] >
name of the document type
pointer to another file
the internal subset of the DTD (optional)
keyword DOCTYPE
DSO Declaration Subset Open
DSC Declaration Subset Close
Internal Subset
Document Entity
<?xml version="1.0"?>
<!DOCTYPE whatnot
[
]>
DOCTYPE declaration includes other declarations in an internal subset
Tags and text: the document

An Example

```
<?xml version="1.0"?>
<!DOCTYPE body
   [
     <!ELEMENT body     (#PCDATA)>
     <!ATTLIST body
       color    CDATA   #IMPLIED>
   ]
>

<body color="blue">Content Goes Here</body>
```

External DTD Declaration


External Subset
Document Entity
<?xml version="1.0"?>
<!DOCTYPE whatnot
SYSTEM "whatnot.dtd" >
DOCTYPE declaration refers to a DTD in a external subset.
a file named: whatnot.dtd
Tags and text: the document

External DTDs

- External DTDs come in two forms:
  - **LOCAL**
  - **PUBLIC**
- Regardless of whether you are using a local or public DTD, to link an external DTD to a document, you must include a DOCTYPE declaration within your XML document just as you should with HTML or XML document.

An Example

```
<?xml version="1.0"?>
<!DOCTYPE address-book SYSTEM "catalog.dtd">

<address-book>
  <contact>
    <last-name>Smith</last-name>
    <first-name>Joe</first-name>
    <phone type="home">408-555-5555</phone>
    <email>Joe.Smith@samplemail.com</email>
  </contact>
</address-book>
```

Specifying a Local DTD

- To specify a DTD on a LOCAL, non-public server, you would use the following format for including the DOCTYPE declaration:

```
<!DOCTYPE root-element SYSTEM “uri-of-dtd”>
```

Specifying a Public DTD

- To specify a DTD on a PUBLIC server that is widely known and advertised, you would use the following format within the DOCTYPE declaration:

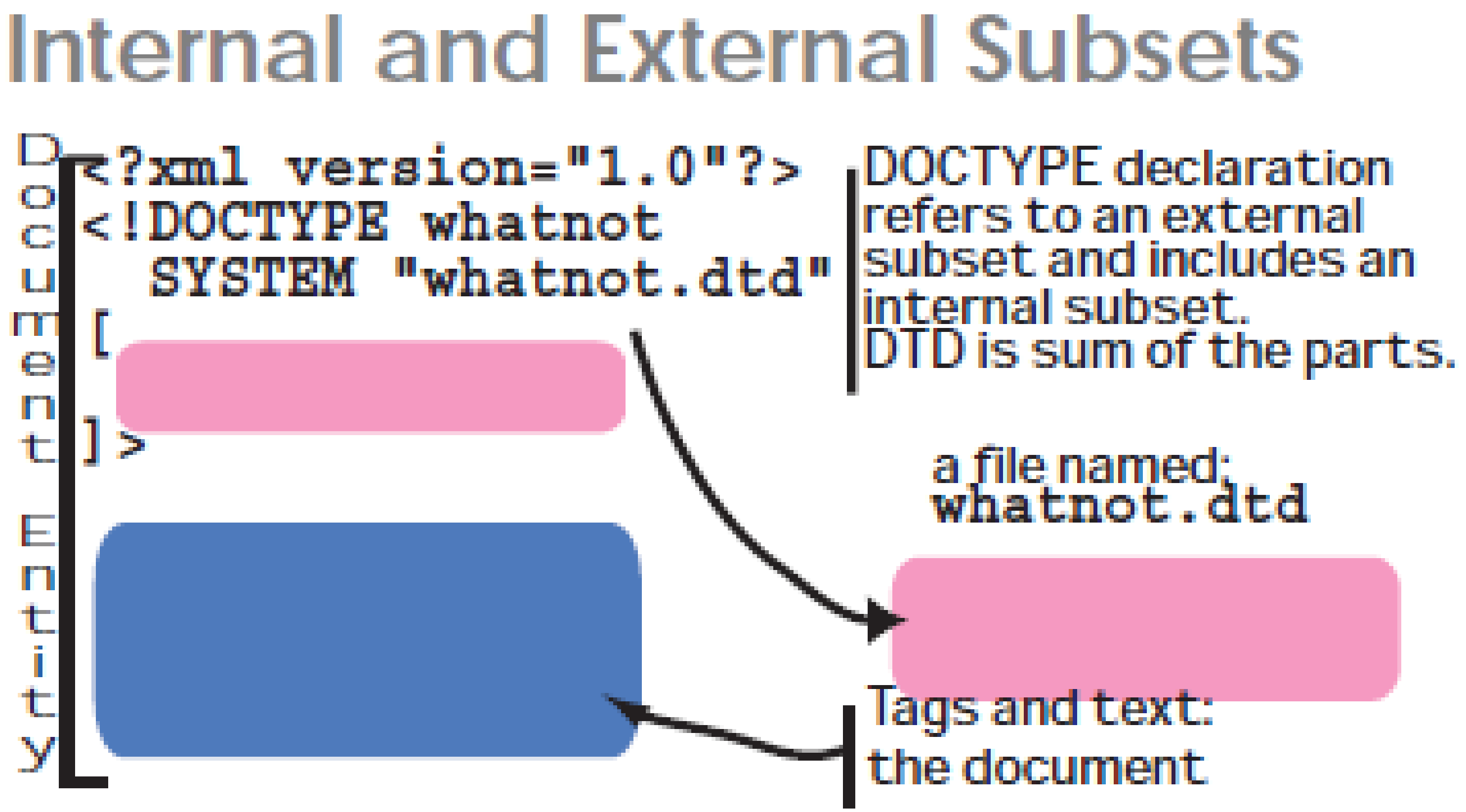
```
<!DOCTYPE root-element PUBLIC “public-identifier”
   “uri-of-dtd”>
```

An Example

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>Content Goes Here</body>
<html>
```

Mixed DTDs Declaration



Internal and External Subsets

```
<?xml version="1.0"?>
<!DOCTYPE whatnot
  SYSTEM "whatnot.dtd"
[

]>
```

Document Entity
DOCTYPE declaration refers to an external subset and includes an internal subset. DTD is sum of the parts.
a file named: whatnot.dtd
Tags and text: the document

An Example

```
<?xml version=“1.0”?>
<!DOCTYPE address-book SYSTEM “catalog.dtd”
  [
    <!ELEMENT phone (home | work)>
    <!ELEMENT home   (#PCDATA)>
    <!ELEMENT work   (#PCDATA)>
  ]
>
<address-book>
  <contact>
    <last-name>Smith</last-name>
    <first-name>Joe</first-name>
    <phone><home>408-555-5555</home></phone>
  </contact>
</address-book>
```

An Example

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
  [
    <!ELEMENT body    (#PCDATA)>
    <!ATTLIST body
      color    CDATA  #IMPLIED>
  ]
>
<html>
  <head>
    <title>Sample Title</title>
  </head>
  <body color="blue">Content Goes Here</body>
</html>
```

The Building Blocks of XML Documents

- Elements
- Attributes

Declaring Elements

- Defining elements within a DTD is done using an `<!ELEMENT>` declaration.

```
<!ELEMENT element-name category>
    or
<!ELEMENT element-name (element-content)>
```

An Example

```
<!ELEMENT   catalog       element_content>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>

  <!-- element content goes here -->

</catalog>
```

← XML File

What an `<!ELEMENT>` Can Contain

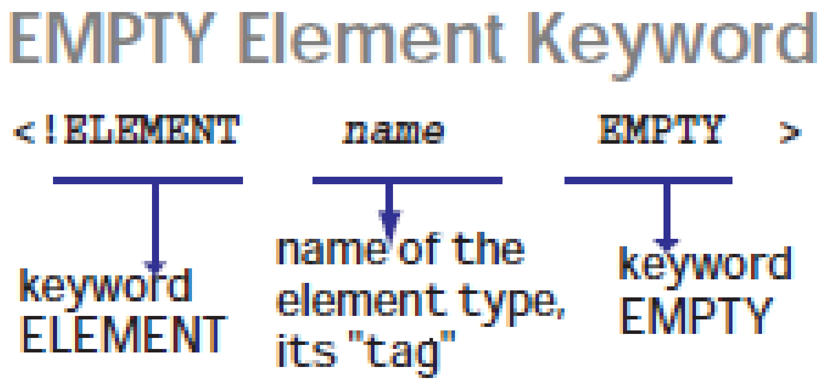
- An `<!ELEMENT>` declaration can contain several different types of content which include the following:

  - **`EMPTY.`**
  - **`PCDATA.`**
  - **`ANY.`**
  - **Children Elements**
  - **Mixed Content**

`EMPTY`

- `<!ELEMENT>` declarations that include the `EMPTY` value allow us to create empty elements within our xml.

- **The word `EMPTY` must be entered in uppercase as it is case-sensitive.**



An Example

```
<!ELEMENT   catalog       EMPTY>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog/>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog></catalog>
```

← XML File

Elements with Parsed Character Data

- Elements with only parsed character data are declared with #PCDATA
- `PCDATA` **is text that will be parsed by a parser. Tags inside the text will treated as markup and entities will be expanded**
- The word `PCDATA` must be enclosed in parenthesis with a preceding `'#'` and entered in uppercase as it is case-sensitive.

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

An Example

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

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog> Household Plants </catalog>
```

← XML File

ANY

- `<!ELEMENT>` declarations that include the value `ANY` allow us include any type of parsable content, including text and other elements, in our elements within our XML instance file.

- **The word `ANY` must be entered in uppercase as it is case-sensitive.**

```
<!ELEMENT   element_name     ANY>
```

An Example

```
<!ELEMENT   catalog       ANY>
<!ELEMENT   xyz       (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <xyz> Catalog Content </xyz>
</catalog>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  Household Plants
</catalog>
```

← XML File

Children Elements

- As mentioned before, the `<!ELEMENT>` declarations can contain children elements allowing for the nesting of elements within a document.

- **Children elements can be added in a variety of ways including adding a single element or multiple elements and optional elements.**

A Single Child Element

- `<!ELEMENT>` declarations that include other elements as values to allow us to specify children elements within our xml.

- **Children elements should be listed after the parent `<!ELEMENT>`.**

```
<!ELEMENT   element_name      (child_element)>
```

An Example

```
<!ELEMENT   catalog        (card)>
<!ELEMENT   card           (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information </card>
</catalog>
```

← XML File

Declaring Multiple Children

- Multiple children elements can be specified within an `<!ELEMENT>` declarations by separating them by commas (,) within parenthesis.

```
<!ELEMENT element_name (child_element,child_element)>
```

An Example

```
<!ELEMENT    catalog        (department,card)>
<!ELEMENT    department     (#PCDATA)>
<!ELEMENT    card           (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <department> Department Name </department>
  <card> Card Information </card>
</catalog>
```

← XML File

Optional Children Elements

- `<!ELEMENT>` declarations that include **zero or one** of a single child element as values can be specified by placing a **question mark** (?) immediately after the child element name.

```
<!ELEMENT   element_name     (child_element?)>
```

An Example of Declaring Zero or One Occurrences of an Element

```
<!ELEMENT    catalog          (card?)>
<!ELEMENT    card             (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog> </catalog>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information </card>
</catalog>
```

An Example

```
<!ELEMENT    class        (room?)>
<|ELEMENT    room         (seats)>
<|ELEMENT    seats        (seat)>
<!ELEMENT    seat         (#PCDATA)>
```

0 or 1 room

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
    <seats>
      <seat>1</seat>
    </seats>
  </room>
</class>
```

Zero or More Children Elements

- `<!ELEMENT>` declarations that include **zero or more** of a single child element as values can be specified by placing a **splat** (*) immediately after the child element name.

```
<!ELEMENT   element_name      (child_element*)>
```

An Example

```
<!ELEMENT    catalog         (card*)>
<!ELEMENT    card            (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog></catalog>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information 1 </card>
  <card> Card Information 2 </card>
</catalog>
```

← XML File

An Example

```
<!ELEMENT   class       (room?)>
<|ELEMENT   room        (seats?)>
<|ELEMENT   seats       (seat*)>
<!ELEMENT   seat        (#PCDATA)>
```

0 or many seats

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
    <seats></seats>
  </room>
</class>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
    <seats/>
  </room>
</class>
```

An Example

```
<!ELEMENT   class    (room?)>
<|ELEMENT   room     (seats?)>
<|ELEMENT   seats    (seat*)>
<!ELEMENT   seat     (#PCDATA)>
```

0 or many seats

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
   <seats>
     <seat>1</seat>
     <seat>2></seat>
   </seats>
  </room>
</class>
```

One Or More Children Elements

- `<!ELEMENT>` declarations that include **one or more** of a single child element as values can be specified by placing a **plus sign** (+) immediately after the child element name.

```
<!ELEMENT   element_name     (child_element+)>
```

An Example

```
<!ELEMENT    catalog        (card+)>
<!ELEMENT    card           (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information </card>
</catalog>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information 1 </card>
  <card> Card Information 2 </card>
</catalog>
```

← XML File

An Example

```
<!ELEMENT    class      (room?)>
<|ELEMENT    room       (seats?)>
<|ELEMENT    seats      (seat+)>
<!ELEMENT    seat       (#PCDATA)>
```

1 or many seats

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
    <seats>
      <seat>1</seat>
    </seats>
  </room>
</class>
```

An Example

```
<!ELEMENT    class       (room?)>
<|ELEMENT    room        (seats?)>
<|ELEMENT    seats       (seat+)>
<!ELEMENT    seat        (#PCDATA)>
```

1 or many seats

```
<?xml version=“1.0” encoding=“UTF-8”?>
<!DOCTYPE class SYSTEM “class.dtd”>
<class>
  <room>
   <seats>
     <seat>1</seat>
     <seat>2></seat>
   </seats>
  </room>
</class>
```

← **XML File**

An Example

```
<!ELEMENT   class      (room?)>
<|ELEMENT   room       (seats?)>
<|ELEMENT   seats      (seat+)>
<!ELEMENT   seat       (#PCDATA)>
```

1 or many seats

```
<?xml version=“1.0” encoding=“UTF-8”?>
<!DOCTYPE class SYSTEM “class.dtd”>
<class>
  <room>
    <seats></seats>
  </room>
</class>
```

```
<?xml version=“1.0” encoding=“UTF-8”?>
<!DOCTYPE class SYSTEM “class.dtd”>
<class>
  <room>
    <seats/>
  </room>
</class>
```

Invalid-
no <set>
elements

An Example

```
<!ELEMENT   class      (windows?, room, door+)>
<|ELEMENT   room       (seats*)>
<|ELEMENT   seats      (seat+)>
<!ELEMENT   seat       (#PCDATA)>
<!ELEMENT   windows    (#PCDATA)>
<!ELEMENT   door       (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE class SYSTEM "class.dtd">
<class>
  <room>
   <seats>
     <seat>1</seat>
     <seat>2></seat>
   </seats>
  </room>
  <door>1</door>
</class>
```

← XML File

Declaring A Choice of Children

- When we have a choice of two or more children elements within an `<!ELEMENT>` declaration, they must be enclosed in parenthesis and must be separated by a pipe (|).

```
<!ELEMENT element_name (child_element|child_element)>
```

```
<!ELEMENT element_name (child_element,
                       (child_element|child_element),
                        child_element)>
```

An Example

```
<!ELEMENT   catalog        (card | cards)>
<!ELEMENT   cards          (card+)>
<!ELEMENT   card           (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information </card>
</catalog>
```

An Example

```
<!ELEMENT    catalog        (card | cards)>
<!ELEMENT    cards          (card+)>
<!ELEMENT    card           (#PCDATA)>
```

← DTD File

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog>
  <cards>
    <card> Card Information </card>
  </cards>
</catalog>
```

← XML File

An Example

```
<!ELEMENT   catalog       (card | cards)>
<!ELEMENT   cards         (card+)>
<!ELEMENT   card          (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <card> Card Information </card>
  <cards>
    <card> Card Information </card>
  </cards>
</catalog>
```

← XML File

An Example

```
<!ELEMENT   class     ((windows|lights), room, door+)>
<|ELEMENT   room      (seats*)>
<|ELEMENT   seats     (seat+)>
<!ELEMENT   seat      (#PCDATA)>
<!ELEMENT   lights    (#PCDATA)>
<!ELEMENT   windows   (#PCDATA)>
<!ELEMENT   door      (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE class SYSTEM "class.dtd">
<class>
  <windows>2</windows>
  <room>
   <seats><seat>1</seat></seats>
  </room>
  <door>north</door>
</class>
```

← XML File

An Example

```
<!ELEMENT   class      ((windows|lights), room, door+)>
<|ELEMENT   room       (seats*)>
<|ELEMENT   seats      (seat+)>
<!ELEMENT   seat       (#PCDATA)>
<!ELEMENT   lights     (#PCDATA)>
<!ELEMENT   windows    (#PCDATA)>
<!ELEMENT   door       (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE class SYSTEM "class.dtd">
<class>
  <lights>2</lights>
  <room>
   <seats><seat>1</seat></seats>
  </room>
  <door>north</door>
</class>
```

← XML File

Declaring Mixed Content

- Declaring mixed content is like declaring a choice with one exception – the first choice is the `#PCDATA` data type within parenthesis with a * following the closing parenthesis.

```
<!ELEMENT element_name (#PCDATA | child_element)*>
```

An Example

```
<!ELEMENT   class      (#PCDATA|windows|lights|room|door)*>
<|ELEMENT   room       (seats*)>
<|ELEMENT   seats      (seat+)>  0 or many seats
<!ELEMENT   seat       (#PCDATA)>
<!ELEMENT   lights     (#PCDATA)>
<!ELEMENT   windows    (#PCDATA)>
<!ELEMENT   door       (#PCDATA)>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE class SYSTEM "class.dtd">
<class>
  <lights>10</lights>
  <door>north</door>
  <windows>2</windows>
</class>
```

← XML File

An Example

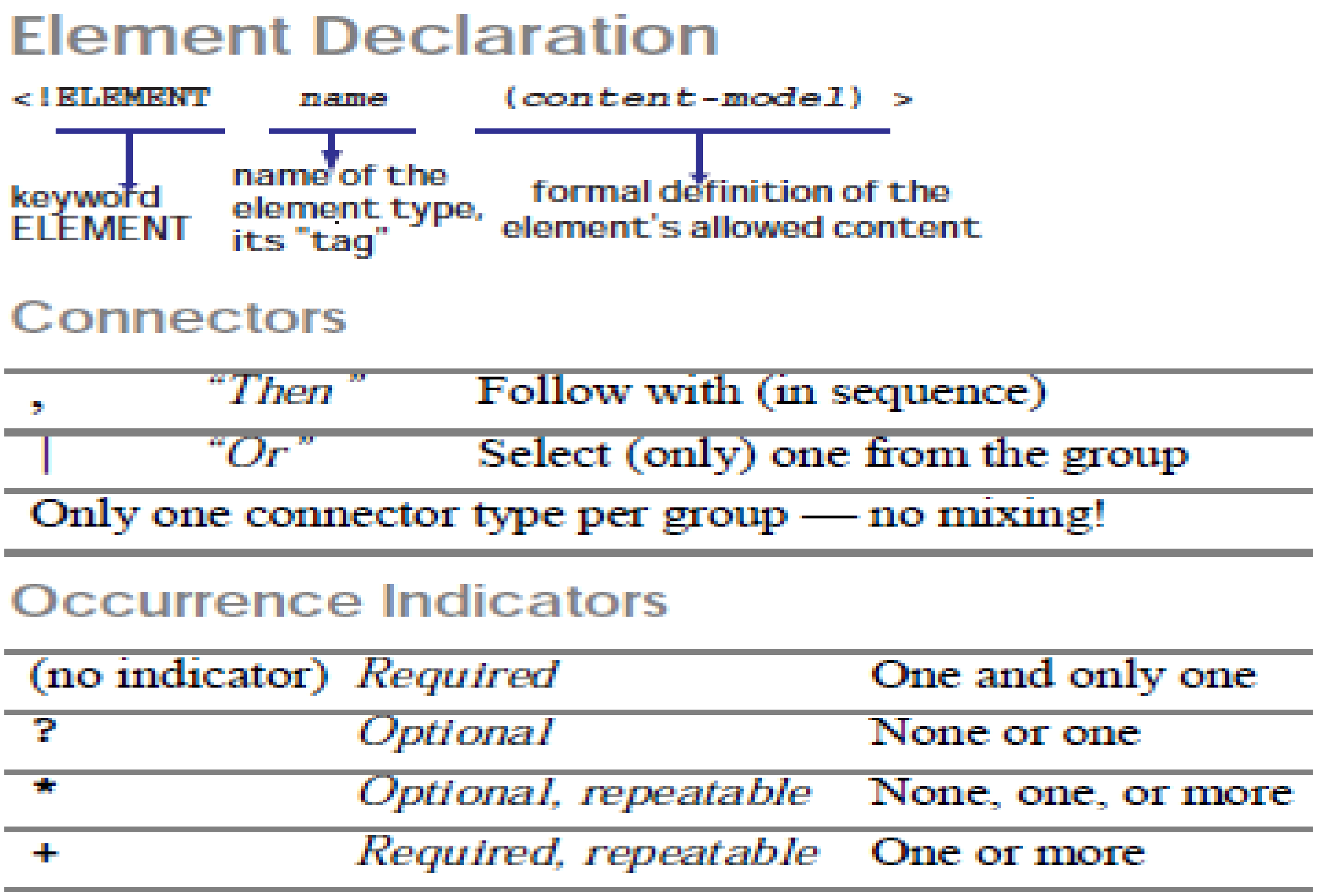
```
<!ELEMENT   class     (#PCDATA|windows|lights|room|door)*>
<|ELEMENT   room      (seats*)>
<|ELEMENT   seats     (seat+)>  0 or many seats
<!ELEMENT   seat      (#PCDATA)>
<!ELEMENT   lights    (#PCDATA)>
<!ELEMENT   windows   (#PCDATA)>
<!ELEMENT   door      (#PCDATA)>
```

← **DTD File**

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE class SYSTEM "class.dtd">
<class>
  <windows>2</windows>,
  <windows>1</windows>,
  <windows>6</windows>
</class>
```

← **XML File**

Summary



Element Declaration

```
<!ELEMENT    name    (content-model) >
```

keyword ELEMENT
name of the element type, its "tag"
formal definition of the element's allowed content

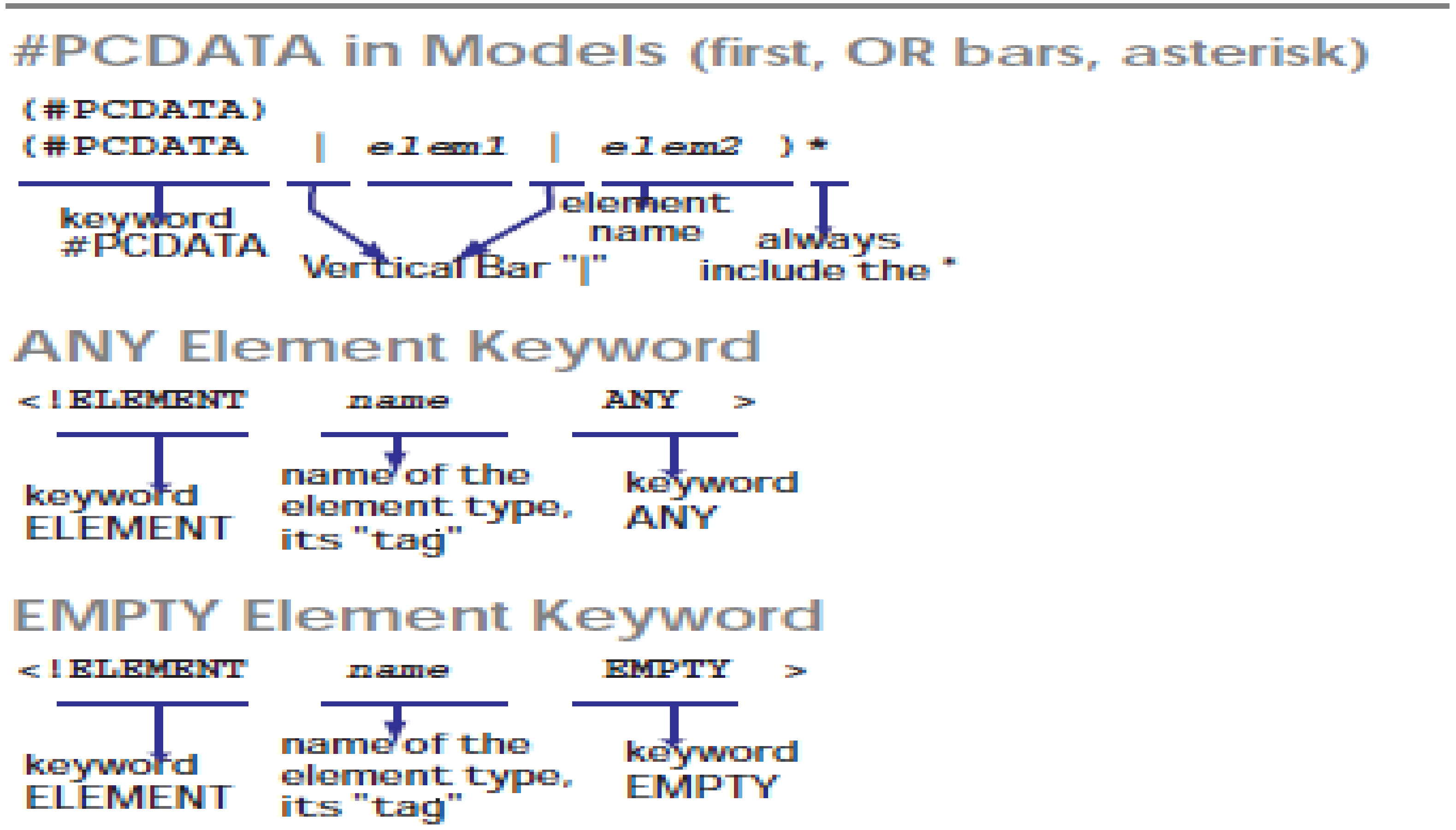
Connectors

,	*"Then"*	Follow with (in sequence)
|	*"Or"*	Select (only) one from the group
Only one connector type per group — no mixing!		

Occurrence Indicators

(no indicator)	*Required*	One and only one
?	*Optional*	None or one
*	*Optional, repeatable*	None, one, or more
+	*Required, repeatable*	One or more

Summary


#PCDATA in Models (first, OR bars, asterisk)
(#PCDATA)
(#PCDATA | elem1 | elem2)*
keyword #PCDATA
Vertical Bar "|"
element name
always include the *
ANY Element Keyword
<!ELEMENT name ANY >
keyword ELEMENT
name of the element type, its "tag"
keyword ANY
EMPTY Element Keyword
<!ELEMENT name EMPTY >
keyword ELEMENT
name of the element type, its "tag"
keyword EMPTY

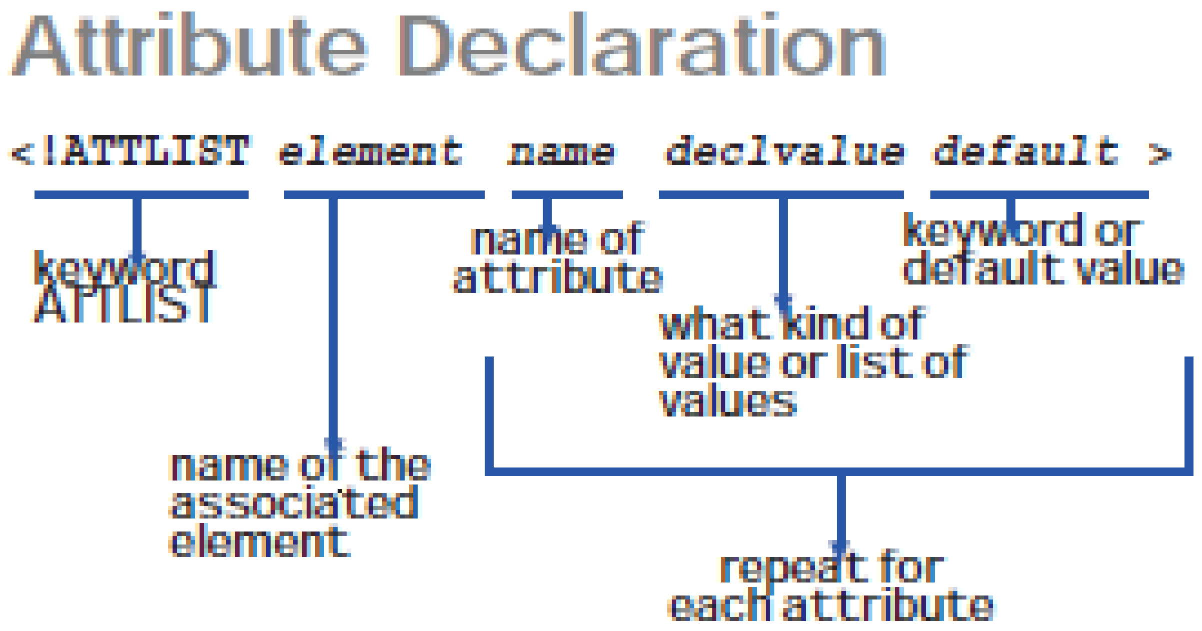
Attributes

- Defining attributes within a DTD is done using an `<!ATTLIST>` declaration.

- **`<!ATTLIST>` declarations are composed of several parts including the element name that the attribute list belongs to, the name of the attribute, the type of content it can contain and how it is required.**

```
<!ATTLIST         element_name
  attribute_name  attribute_type  default_value
>
```

Attributes


Attribute Declaration
<!ATTLIST element name declvalue default >
keyword ATTLIST
name of the associated element
name of attribute
what kind of value or list of values
keyword or default value
repeat for each attribute

More About Attributes

- When including the `<!ATTLIST>` declaration, it must be placed IMMEDIATELY after the `<!ELEMENT>` declaration that it is associated with. Children `<!ELEMENT>`s will follow the `<!ATTLIST>` declaration.

- **Multiple attributes for one `<!ELEMENT>` declaration are contained within the same `<!ATTLIST>` declaration.**

```
<!ELEMENT         element_name    element_contents>
<!ATTLIST         element_name
  attribute_name1 attribute_type  default_value
  attribute_name2 attribute_type  default_value
>
```

Values of Attributes

- An `<!ATTLIST>` declaration can be defined as having one of four different value which include the following:

  - ***`value`***
    – This is the DTD defined default value.
  - **`#REQUIRED`**
    – An attribute with this value must be included.
  - **`#IMPLIED`**
    – An attribute with this value is optional.
  - **`#FIXED`** ***`value`***
    – This defines fixed values for the given attribute.

An Example using #REQUIRED

```
<!ELEMENT   catalog   EMPTY>
<!ATTLIST   catalog
            catid     CDATA     #REQUIRED
>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog catid="D123"/>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog/>
```

This is invalid since the attribute is missing.

An Example using #IMPLIED

```
<!ELEMENT   catalog   EMPTY>
<!ATTLIST   catalog
            catid     CDATA     #IMPLIED
>
```

← DTD File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog catid="D123"/>
```

← XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog/>
```

← XML File

An Example using #FIXED Values

```
<!ELEMENT   catalog   EMPTY>
<!ATTLIST   catalog
            catid     CDATA     #FIXED “D123”
>
```

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D123”/>
```

← XML File

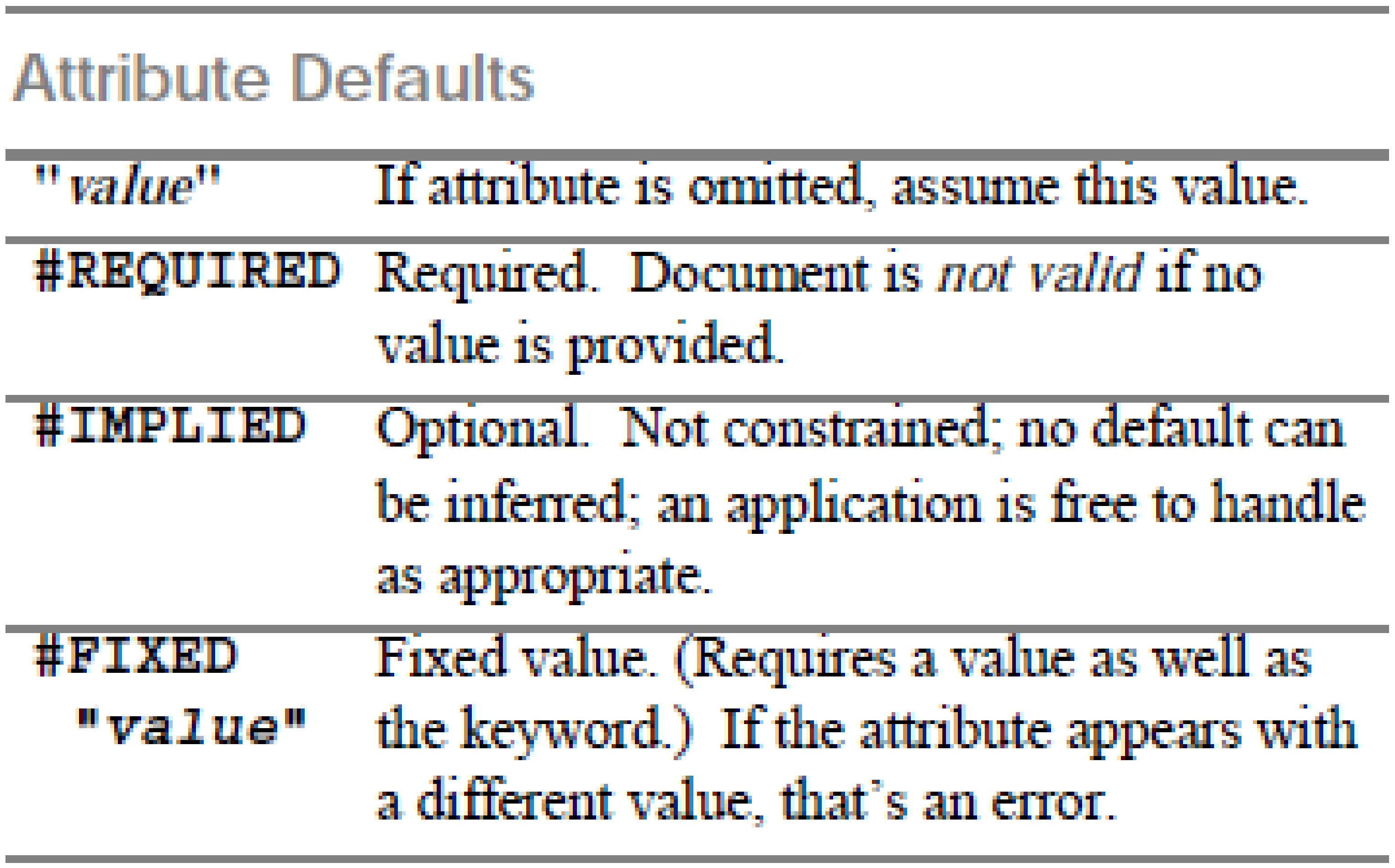
```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog/>
```

Attribute is missing.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D456”/>
```

Invalid value used.

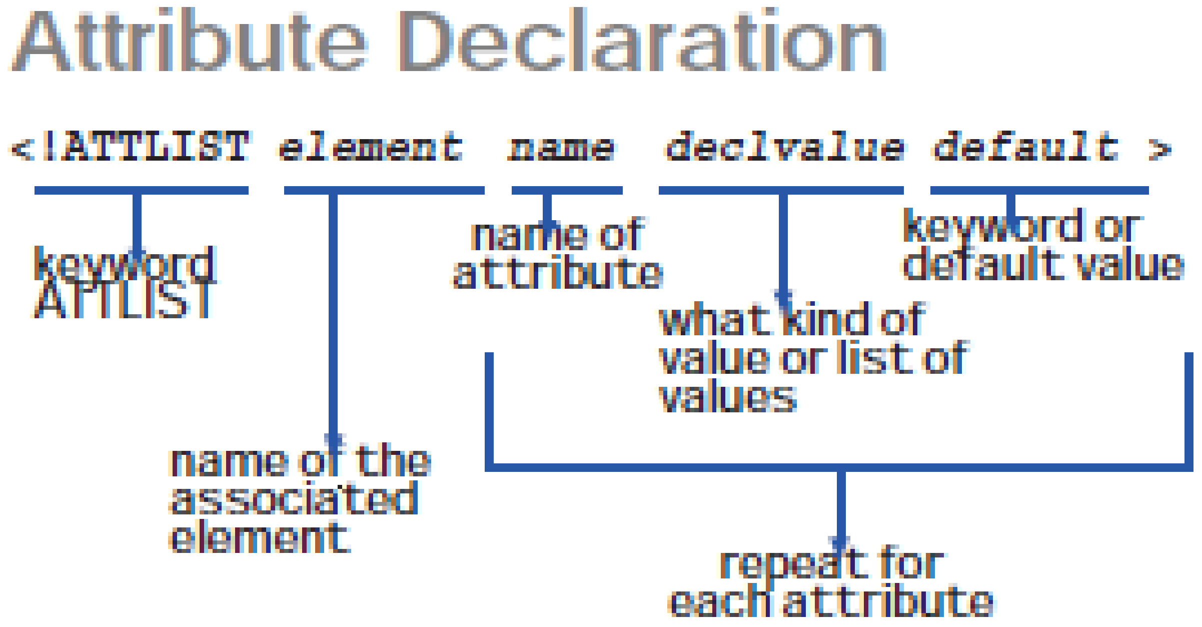
Summary



Attribute Defaults

"*value*"	If attribute is omitted, assume this value.
`#REQUIRED`	Required. Document is *not valid* if no value is provided.
`#IMPLIED`	Optional. Not constrained; no default can be inferred; an application is free to handle as appropriate.
`#FIXED "value"`	Fixed value. (Requires a value as well as the keyword.) If the attribute appears with a different value, that's an error.

Attributes


Attribute Declaration
<!ATTLIST element name declvalue default >
keyword ATTLIST
name of the associated element
name of attribute
what kind of value or list of values
keyword or default value
repeat for each attribute

Declaring Attributes

- An `<!ATTLIST>` declaration can be defined as one of several different types which include the following:

   - **`CDATA`**
      - The value is character data
   - **Enumerated Types**

   The value must be one from an enumerated list

   - **ID**

     The value is a unique id

   Unique value must begin with a Non-Integer value

An Example using CDATA

```
<!ELEMENT   catalog   EMPTY>
<!ATTLIST   catalog
            catid     CDATA     “D123”
>
```

← DTD File

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D123”/>
```

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D456”/>
```

An Example using Enumerated Types

```
<!ELEMENT    catalog    EMPTY>
<!ATTLIST    catalog
             catid       (D123|D234|D345)      “D123”
>
```

← DTD File

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog/>
```

Attribute and value is assumed.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D123”/>
```

Bad Example.

```
<?xml version=“1.0” encoding=“UTF-8”?>
<DOCTYPE catalog SYSTEM “catalog.dtd”>
<catalog catid=“D456”/>
```

Summary

```
<!ATTLIST book
publisher CDATA #IMPLIED
reseller CDATA  #FIXED "MyStore"
ISBN  ID  #REQUIRED
inPrint  (yes|no) "yes" >
```

Disadvantages of DTD

- **DTD has its own syntax.**
- **Precise number of element repetitions can’t be achieved.**
- **Limited number of data types.**
- **Only 1 DTD can be referenced from within the XML document.**

Thank you !