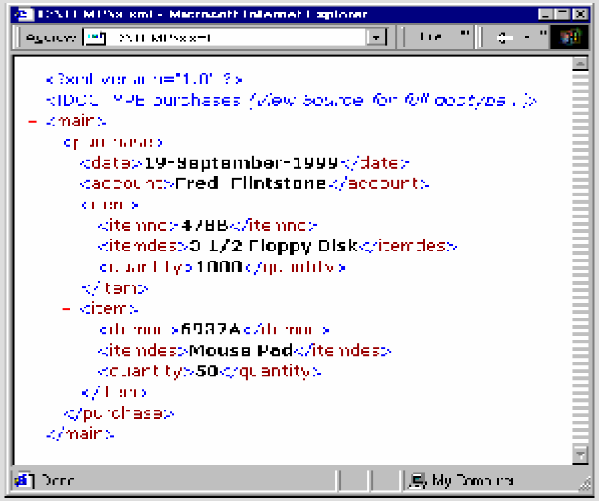
XML Structure and Syntax

5-2

November 3, 2003

A Simple Example



```
<?xml version="1.0" ?>
<!DOCTYPE purchases (View Source for full doctype...)>
- <main>
  <purchase>
    <date>19-September-1999</date>
    <account>Fred Flintstone</account>
    <item>
      <itemno>478B</itemno>
      <itemdes>3 1/2 Floppy Disk</itemdes>
      <quantity>1000</quantity>
    </item>
  - <item>
      <itemno>6937A</itemno>
      <itemdes>Mouse Pad</itemdes>
      <quantity>50</quantity>
    </item>
  </purchase>
</main>
```

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

Document Type *Definition* specified by the Document Type *Declaration*

```
<!DOCTYPE main [
<!ELEMENT main (purchase)*>
<!ELEMENT purchase (date, account, item+)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT account (#PCDATA)>
<!ELEMENT item (itemno, itemdes, quantity)>
<!ELEMENT itemno (#PCDATA)>
<!ELEMENT itemdes (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
]>
```

XML Marked-Up Data

```
<main>
  <purchase>
     <date>19-September-1999</date>
     <account>Fred_Flintstone</account>
     <item>
        <itemno>478B</itemno>
        <itemdes>3 1/2 Floppy Disk</itemdes>
        <quantity>1000</quantity>
     </item>
     <item>
        <itemno>6937A</itemno>
        <itemdes>Mouse Pad</itemdes>
        <quantity>50</quantity>
     </item>
  </purchase>
</main>
```

Some Key Things…

- **Extensibility**
  - not limited to a pre-defined set of tags: "…allows an author to define a particular structure…by defining the parts (tags) that fit that structure."
- **Everything is Textual**
  - gives readability—makes life a LOT easier for the developer; makes interoperation smoother; etc.
  - can be efficient because text is usually highly compressible
- **Is the foundation for a whole lot of new work**
  - interoperability has previously been approached in a closed fashion
    - *mostly just groups of vendors (sometimes!) agreeing to support each other's file formats, etc.*
  - now being tackled by the standards community

- **XML document composed of:**
  - declarations, elements, entities, processing instructions and comments
    - *some are optional, some required*
    - *an important aspect is the possibility of creating* self-describing *documents*
  - may also contain processing code

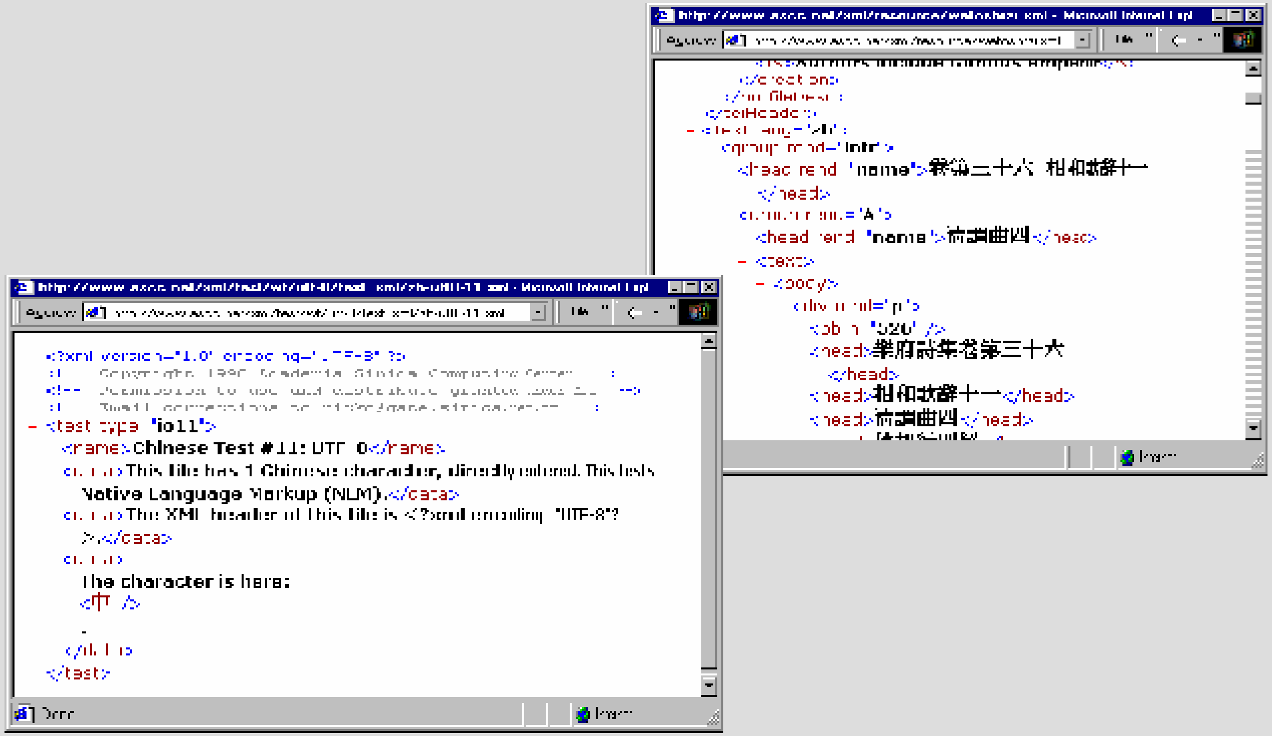
XML Structure…

- **Two aspects:**
  - logical
    - *defined by the* Document Type Definition *(DTD): what to include, what relationships pertain between the various parts*
  - physical
    - *the actual data being tagged*
    - *correspondence with the logical structure can be checked*
  - an analogy…
    - *consider a block of flats: the plan defines the block's logical structure while "Young Miss Jones over in Flat 32B" relates to the physical makeup. Allows validation, thus: "How come Old Mr. Smith spent the night in Young Miss Jones' Flat?"*

- **Tags are defined according to need**
  - accounts for XML's extensibility
- **XML can use Unicode**

```
<?xml version="1.0" encoding= "UTF-8" ?>
```

  - can deal with multi-byte character sets



```
<?xml version="1.0" encoding="UTF-8" ?>
- <test type="ioll">
    <name>Chinese Test #11: UTF 8</name>
    <data>This file has 1 Chinese character, directly entered. This tests
      Native Language Markup (NLM).</data>
    <data>The XML header of this file is <?xml encoding="UTF-8"?
      >.</data>
    <data>
      The character is here:
      <巾 />
      .
    </data>
  </test>
```

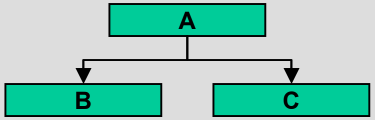
```
      </creation>
    </profileDesc>
  </teiHeader>
- <text lang='zh'>
    <group rend='intr'>
      <head rend='name'>卷第三十六 相和歌辭十一
        </head>
      <group n='A'>
        <head rend='name'>清調曲四</head>
      - <text>
        - <body>
            <div n='p'>
              <pb n='520' />
              <head>樂府詩集卷第三十六
                </head>
              <head>相和歌辭十一</head>
              <head>清調曲四</head>
```

Document Tree

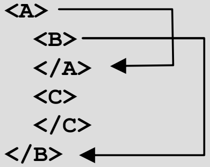
- **Elements define a tree structure**
  - must be correctly nested: *perfect nesting* is required

✓

```
<A>
  <B>
  </B>
  <C>
  </C>
</A>
```



✗

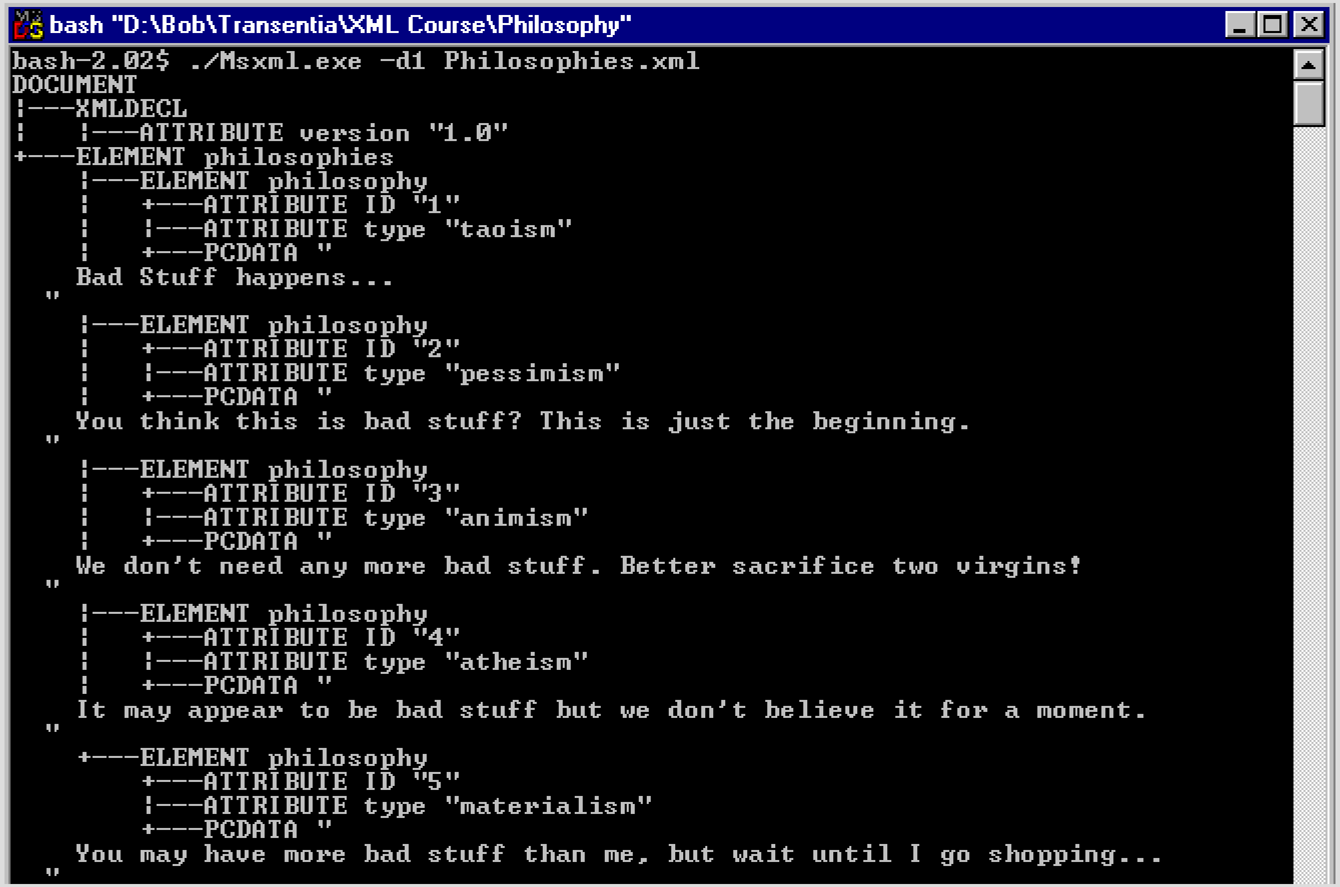


```
<A>
  <B>
  </A>
  <C>
  </C>
</B>
```

  - note the single document root
    - *everything must be contained within this*

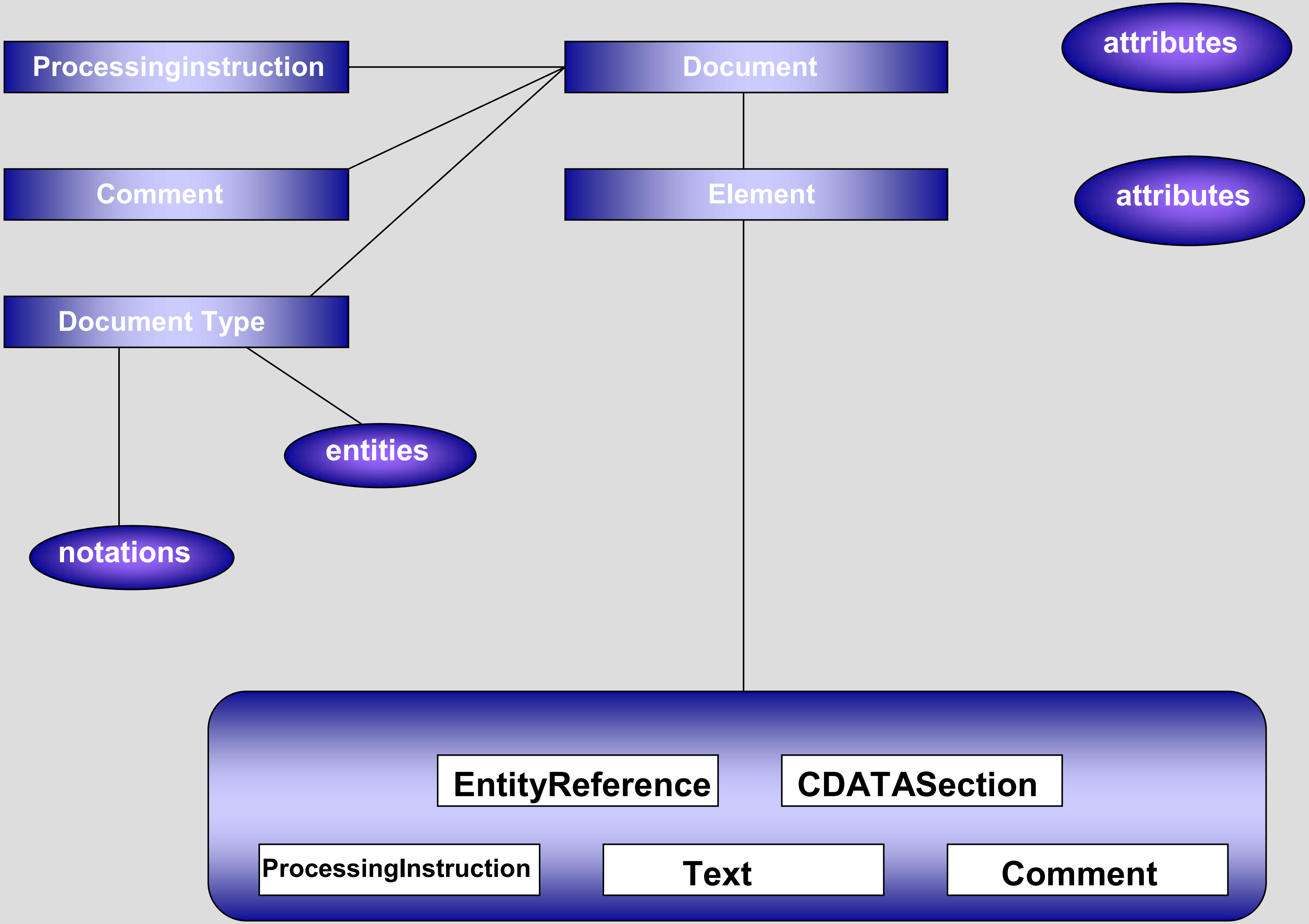
More Tree...

```
<?xml version="1.0"?>
<philosophies>
  <philosophy ID="1" type="taoism">
    Bad Stuff happens...
  </philosophy>
  <philosophy ID="2" type="pessimism">
    You think this is bad stuff? This is just the beginning.
  </philosophy>
  <philosophy ID="3" type="animism">
    We don't need any more bad stuff. Better sacrifice two virgins!
  </philosophy>
  <philosophy ID="4" type="atheism">
    It may appear to be bad stuff but we don't believe it for a moment.
  </philosophy>
  <philosophy ID="5" type="materialism">
    You may have more bad stuff than me, but wait until I go shopping...
  </philosophy>
</philosophies>
```



```
bash-2.02$ ./Msxml.exe -d1 Philosophies.xml
DOCUMENT
¦---XMLDECL
¦   ¦---ATTRIBUTE version "1.0"
+---ELEMENT philosophies
    ¦---ELEMENT philosophy
    ¦   +---ATTRIBUTE ID "1"
    ¦   ¦---ATTRIBUTE type "taoism"
    ¦   +---PCDATA "
    Bad Stuff happens...
  "
    ¦---ELEMENT philosophy
    ¦   +---ATTRIBUTE ID "2"
    ¦   ¦---ATTRIBUTE type "pessimism"
    ¦   +---PCDATA "
    You think this is bad stuff? This is just the beginning.
  "
    ¦---ELEMENT philosophy
    ¦   +---ATTRIBUTE ID "3"
    ¦   ¦---ATTRIBUTE type "animism"
    ¦   +---PCDATA "
    We don't need any more bad stuff. Better sacrifice two virgins!
  "
    ¦---ELEMENT philosophy
    ¦   +---ATTRIBUTE ID "4"
    ¦   ¦---ATTRIBUTE type "atheism"
    ¦   +---PCDATA "
    It may appear to be bad stuff but we don't believe it for a moment.
  "
    +---ELEMENT philosophy
        +---ATTRIBUTE ID "5"
        ¦---ATTRIBUTE type "materialism"
        +---PCDATA "
    You may have more bad stuff than me, but wait until I go shopping...
  "
```

Structure



Processinginstruction

Document

attributes

Comment

Element

attributes

Document Type

entities

notations

EntityReference

CDATASection

ProcessingInstruction

Text

Comment

Syntax: Elements

- **Elements**
  - containers for data
  - may be empty
    - *special syntax*

```
<TAG></TAG>          <TAG />
```

  - may possess associated attributes
  - naming rules
    - *case sensitive*
    - *names beginning with xml (in any combination of case) are reserved*
    - *the use of a colon in a name should be avoided*
      - reserved for experimentation

Attribute

Container Elements

```
<EVENT TYPE="Birthday">
  <DATE>
    25-04-1999
  </DATE>
  <MESSAGE>
    The better-half's 40th…FORGET THIS AND DIE!!!
  </MESSAGE>
</EVENT>
```

✓

```
<_>
<fred>
<FreD99>
<_fred>
<a.long.name>
<another_long-name>
```

✗

```
<a bad name>
<$5>
<help!>
<:fred>
```

- **Whitespace is important in XML**
  - within a mixed content model (a content model that allows both elements and text nodes), XML considers white space to be significant and preserves it. In other content models, white space is ignored.
  - the special attribute *xml:space* may be attached to an element to tell the processing application what the XML document wants it to do with whitespace
    - *only possible values are "default" and "preserve"*
  - line breaks in non-markup data are also preserved and are signalled to the application
    - *as a single newline character: '**#xA**'*

```
<?xml version="1.0" ?>
<!DOCTYPE HAIKU [
<!ELEMENT HAIKU (#PCDATA)>
<!ATTLIST HAIKU xml:space (default|preserve) 'preserve'>
]>
<HAIKU>
I'm sorry, there's -- um --
    insufficient -- what's-it-called?
        The term eludes me ...
-- Owen Mathews
</HAIKU>
```

Syntax: Attributes

- **Attributes**
  - provides a way of associating values to an element *without that value actually being part of the element*

```
<SHOE ID="99864127-A">
```

  - *the value part* must *be enclosed in quotes*
    - either double or single can be used
    - treated as plain text—not parsed for further markup
      - *still must match encoding appropriate to host element*
  - *xml:lang is a reserved attribute*
    - identifies the human language in which the element was written

```
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
```

- **Which to choose?**
  - one of the most common questions…
  - attributes and elements seem to be interchangeable from the modelling point of view
- **No hard-and-fast rule**
  - next couple of slides present guidelines…
  - a few "rules of thumb":

"Generally speaking, when human beings are expected to create or work with XML documents, it is better to use elements than attributes."

elements "push" the program along, while attributes are "pulled" in…

The technical case against attributes is very strong… "Two ways of representing the simplest of data (a name/value pair) has caused a fracture that has propagated through the DOM, DTDs, namespaces, queries, schemas, etc. and the higher it goes the more problems it is causing." The only argument for keeping attributes seems to be legalistic precedence, that is, "we made that mistake so long ago that we cannot fix it now."

Forced Choices

- **A few guidelines:**
  - the data contains substructures
    - *model as an element: attributes take only simple strings*
  - the data contains multiple lines
    - *attributes should be simple and short or they become unreadable/unusable*
  - the data changes frequently
    - *when the data will be frequently modified, especially by the end user, then model as an element: XML editors usually make it easy to find and modify element data*
  - the data is confined to a small number of fixed choices
    - *makes sense to use an attribute, which can be prevented from taking on any value that is not in the predefined list*

- **A few more heuristics:**
  - visibility
    - *if the data is intended to be shown to an end user then it should be an element but if the information guides processing but is never displayed, then it may be better as an attribute*
  - consumer / provider
    - *if data is entered by a human, it may be best as an element*
  - container vs. contents
    - *another way of thinking about elements and attributes is to think of an element as a container. Characteristics of the* container itself *correspond to attributes.*

YAML

- **Some people think that XML is too complex…**
  - various alternatives exist
    - *YAML is one example that is oriented towards PERL programmers*
      - With maps (%), lists (@) and scalars ($)
    - *Aims for simplicity and better data density*

```
buyer    : %
    address     : %
       city       : Royal Oak
       line one   : 458 Wittigen's Way
       line two   : Suite #292
       postal     : 48046
       state      : MI
    family name : Dumars
    given name  : Chris
date     : 12-JAN-2001
comments :
    Mr. Dumars is frequently gone in the morning so try late afternoon.
delivery : %
    method : UZS Express Overnight
    price  : 45.50
invoice : 00034843
product : @
    %
        desc      : Grade A, Leather Hide Basketball
        id        : BL394D
        price     : 450.00
        quantity  : 4
    %
        desc      : Super Hoop (tm)
        id        : BL4438H
        price     : 2,392.00
        quantity  : 1
tax      : 0.00
total    : 4237.50
```

Syntax: Comments

- **Comments**
  - enable the creator of a document to explain *why* something was done the way it was
    - *or to assert ownership over the data*
    - *or anything else…*

```
<!-- this is a comment -->
```

  - do not nest

```
<!-- this is a comment <!-- this is illegal --> -->
```

  - can only be placed *outside* other markup

```
<TAG <!-- you can’t do this --> >     <TAG> <!-- this is OK -->
```

  - must not contain **--**
    - *keeps XML compatible with SGML*
  - contents can be examined by processor if needed
    - *bad style…*

Syntax: CDATA

- **CDATA**
  - “Character Data”
    - *not marked-up data*
  - an ‘escape’ facility—“leave me alone”
    - *XML’s normal parsing turned off for the CDATA section*

```
<PARA>
  In XML a comment is opened with the sequence <![CDATA[<!--]]>.
</PARA>
<PARA>
  A comment finishes with the sequence <![CDATA[-->]]>.
</PARA>
```

-
  -
    - *one way of embedding code…*
  - obviously, the actual data cannot contain]]>
    - *otherwise the CDATA section would be incorrectly terminated*

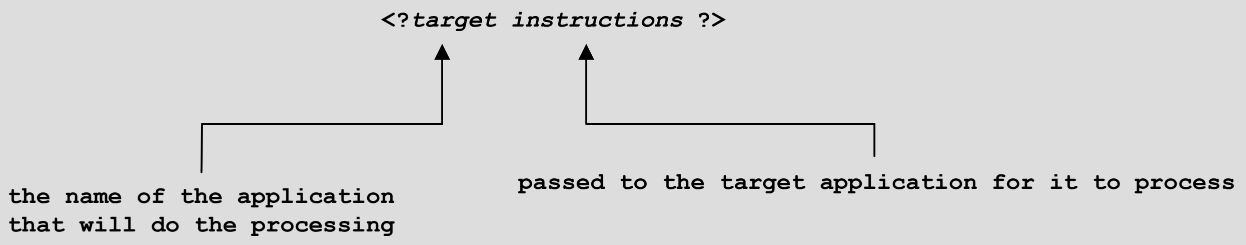
```
<![CDATA[This is left alone $&$(&&&!!! <!—><>>>]]> but this gives errors]]>
```

-
  - overuse is bad style
    - *since the structure of the included data remains opaque*
  - CDATA is used for attribute values

```
<?xml version=“1.0”?>
<PROGRAM lang=“BASIC” xml:space=“preserve”>
10 LET A=10
20 LET B=20
30 IF A <![CDATA[<]]>B THEN PRINT A+B
</PROGRAM>
```

Syntax: PIs

- **Processing Instructions**
  - ignored by XML itself; commands or information is passed straight to the application that is processing the XML data



```
<?target instructions ?>
```

the name of the application that will do the processing

passed to the target application for it to process

  - may need many processing instructions if allowing for many alternative processing environments
    - *a potential (and nasty) 'gotcha'*

```
<PARA>
  this element also contains two processing instructions (to allow for
  two different processing applications)
  <?javascript do something for javascript ?>
  <?perlscript do something equivalent for perlscript ?>
</PARA>
```

  - the target name "xml" (in any combination of cases) is reserved for use by XML itself:

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

Syntax: Entities

- # A sort of macro facility

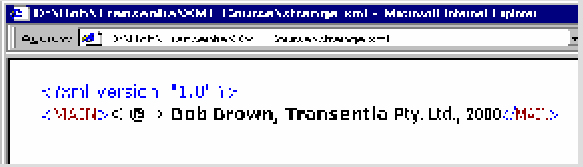
```
<!ENTITY name "Bob Brown">
…
<PARA>Hi there! I am &name;. Who are you?</PARA>
```

- can give a facility similar to C’s #define…
  - *and with the same dangers*
    - it’s possible to construct non-syntactic substitutions, for example
- also provides ability to construct a document from external parts
- there exist a number of predefined entities for key parts of the XML markup language that cannot be directly used as content:

&lt;	&gt;	&amp;	&apos;	&quot;
<	>	&	`	"

- character entities allow arbitrary characters to be represented in the XML text

```
<?xml version="1.0" ?>
<MAIN>
  &lt; &#169; &gt; Bob Brown, Transentia Pty. Ltd., 2000
</MAIN>
```


D:\Work\Transentia\XML Course\strange.xml - Microsoft Internet Explorer
Address

```
<?xml version="1.0" ?>
<MAIN>< © > Bob Brown, Transentia Pty. Ltd., 2000</MAIN>
```

More Entities…

- **Four varieties:**
  - Parsed
    - *used to include "boilerplate" text in a document*

```
<!ENTITY H "Hydrogen">
…
<SENTENCE>
  &H; is the lightest of all atoms.
</SENTENCE>
```

  - *may be internal or external*
    - support for external parsed entities is optional for non-validating parsers
  - Unparsed
    - *used to pass a reference to a piece of external content, along with a description of the type of the content*
      - uses NOTATIONs
  - Internal
  - External

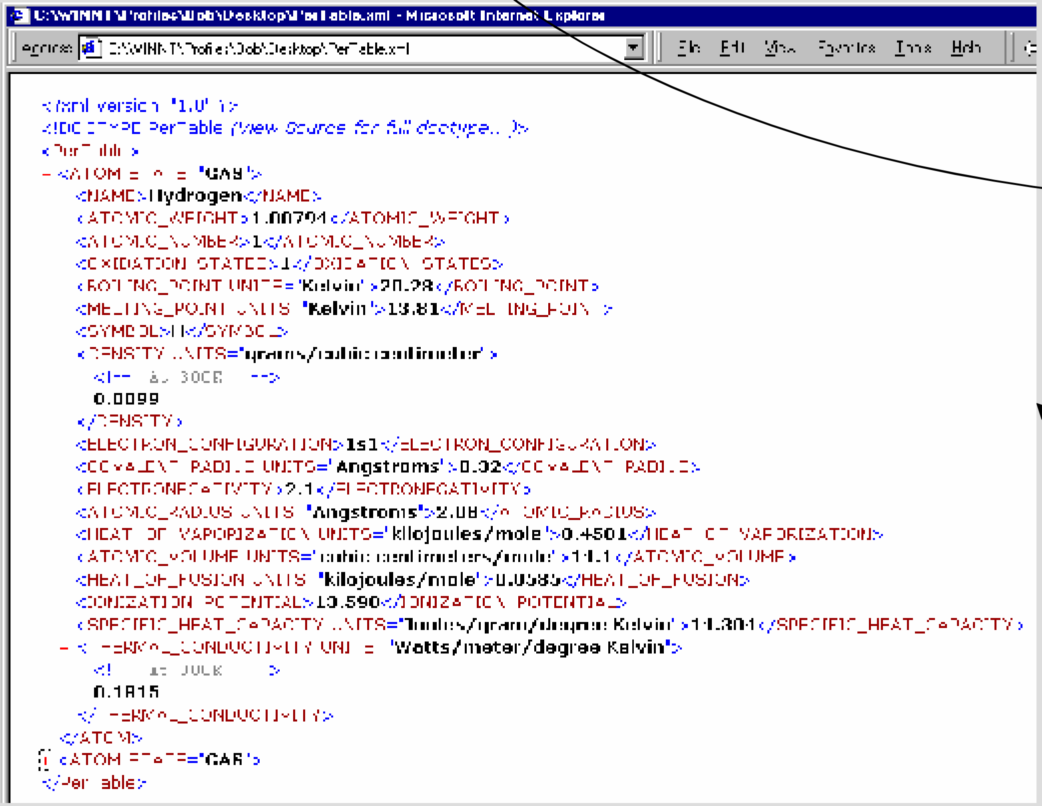
External Entities

```
<ATOM STATE="GAS">
  <NAME>Hydrogen</NAME>
  <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
  <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
  <OXIDATION_STATES>1</OXIDATION_STATES>
  …
</ATOM>
```

```
<ATOM STATE='GAS'>
  <NAME>Helium</NAME>
  <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
  <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
  <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
  <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
  <SYMBOL>He</SYMBOL>
  <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
    0.1785
  </DENSITY>
  <ELECTRON_CONFIGURATION>1s2 </ELECTRON_CONFIGURATION>
  …
  <THERMAL_CONDUCTIVITY UNITS="Watts/meter/degree Kelvin">
  <!-- At 300K -->
    0.152
  </THERMAL_CONDUCTIVITY>
</ATOM>
```

```
<!ENTITY H SYSTEM "h.xml">
<!ENTITY He SYSTEM "He.xml">
```

```
<?xml version="1.0" ?>
<!DOCTYPE PerTable [
<!ENTITY % Elements SYSTEM "Elements.xml">
%Elements;
]>
<PerTable>
  &H;
  &He;
</PerTable>
```


C:\WINNT\Profiles\Bob\Desktop\PerTable.xml - Microsoft Internet Explorer
Address
C:\WINNT\Profiles\Bob\Desktop\PerTable.xml
File
Edit
View
Favorites
Tools
Help

```
<?xml version="1.0" ?>
<!DOCTYPE PerTable (View Source for full doctype...)>
<PerTable>
- <ATOM STATE="GAS">
    <NAME>Hydrogen</NAME>
    <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
    <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
    <OXIDATION_STATES>1</OXIDATION_STATES>
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <SYMBOL>H</SYMBOL>
    <DENSITY UNITS="grams/cubic centimeter">
      <!-- At 300K -->
      0.0099
    </DENSITY>
    <ELECTRON_CONFIGURATION>1s1</ELECTRON_CONFIGURATION>
    <COVALENT_RADIUS UNITS="Angstroms">0.32</COVALENT_RADIUS>
    <ELECTRONEGATIVITY>2.1</ELECTRONEGATIVITY>
    <ATOMIC_RADIUS UNITS="Angstroms">2.08</ATOMIC_RADIUS>
    <HEAT_OF_VAPORIZATION UNITS="kilojoules/mole">0.4501</HEAT_OF_VAPORIZATION>
    <ATOMIC_VOLUME UNITS="cubic centimeters/mole">14.1</ATOMIC_VOLUME>
    <HEAT_OF_FUSION UNITS="kilojoules/mole">0.0585</HEAT_OF_FUSION>
    <IONIZATION_POTENTIAL>13.590</IONIZATION_POTENTIAL>
    <SPECIFIC_HEAT_CAPACITY UNITS="Joules/gram/degree Kelvin">14.304</SPECIFIC_HEAT_CAPACITY>
  - <THERMAL_CONDUCTIVITY UNITS="Watts/meter/degree Kelvin">
      <!-- At 300K -->
      0.1815
    </THERMAL_CONDUCTIVITY>
  </ATOM>
+ <ATOM STATE="GAS">
</PerTable>
```

Syntax: Notation

- **Identify the format of the content of unparsed entities by name**

```
<!ENTITY myImage SYSTEM "image.gif" NDATA GIF>
```

– entity declaration doesn’t tell the processor what to do with the type of data; needs an associated notation declaration:

```
<!NOTATION GIF SYSTEM "Iexplore.exe">
```

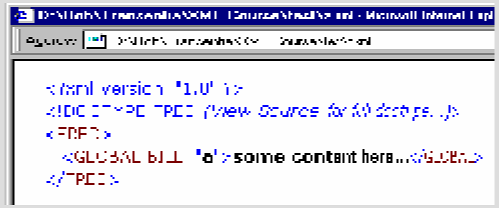
- *note how this incorporates very platform-specific information*
  - – probably only useful on the server side…
  - – also implies versioning problems
  - – may prefer to use something like HTML’s IMG tag:

```
<!ELEMENT slide (image?, title, item*)>
<!ATTLIST slide type (tech | exec | all) #IMPLIED >
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image alt CDATA #IMPLIED
                src CDATA #REQUIRED
                type CDATA "image/gif" >
```

- *or use XLink (if & when it is specified...)*

– myImage is an *unparsed entity*

- **Document Type Definition**
  - XML's information modelling facility
  - specified by the *Document Type Declaration* at the top of the file
  - allows a processing application to determine whether the data contained in an XML is structured correctly:
    - *no missing (required) data/attributes*
    - *no extra (unexpected) data*
    - *relationships between different parts is correct*
  - similar to a database schema
    - *not as thorough, though*
      - being worked on…
  - needs careful development
    - *must ensure that the definition is unambiguous*
      - otherwise processing becomes difficult/inefficient or even incorrect



```
<?xml version="1.0" ?>
<!DOCTYPE FRED (View Source for full doctype...)>
<FRED>
  <GLOBAL BILL="a">some content here...</GLOBAL>
</FRED>
```

```
<?xml version="1.0" ?>
<!DOCTYPE FRED [
<!ELEMENT FRED (GLOBAL)>
<!ELEMENT GLOBAL (#PCDATA)>
<!ATTLIST GLOBAL
  BILL (a|b|c) "c">
]>
<FRED>
  <GLOBAL BILL="a">some content here…</GLOBAL>
</FRED>
```

Why Have A DTD?

- **Well-formed versus valid XML**
  - *DTD allows correspondence between physical and logical structure to be checked*
  - *a* well-formed *document is structurally sound but may contain other errors*
    - missing elements, attributes, etc.; duplicate IDs that should be unique, etc.
    - an XML document without an accompanying DTD can really only be checked for “well formed-ness”
  - *a* valid *document must be well-formed,* plus *the contents of the document must conform to the rules specified in the DTD*
    - only if there is an associated DTD can conformance be checked: *“…unlike HTML, the built-in validity checking of XML allows users to trust the data. Validity checking makes XML appropriate for transactions, electronic commerce and inventory management.”*

Reiterating...

- **The DTD allows:**
  - the document's *structure* to be defined
  - the document's *contents* to be defined
    - *badly! A schema is a much better tool…*

Validity Without DTD

- **It *is* possible to validate a document without requiring DTDs**
  - thus:
    - *all attribute values would have to be given; there could be no assumed default values*
    - *there could be no entity mechanism*
    - *whitespace handling would be very difficult*
      - when to ignore?
  - the probability of unchecked errors being introduced would become very high
    - *the requirement for self-discipline would be too great*
      - *I* am well-disciplined, but are *you*?
        - *:-)*
- **Checking against a DTD doesn't prevent garbage *content*, just lousy *structure*:**

```
<DATE>the owl and the pussycat went to sea…</DATE>
```

More DTD…

- **The Document Type Definition is actually composed from the union of two subsets:**
  - external
    - *included into the XML document being examined/processed by reference*
  - internal
    - *contained within the actual XML document*
      - read before external DTD subset (if one is specified) and so takes precedence
        - “the first one in wins”
  - DTD can be completely internal, completely external or some mixture

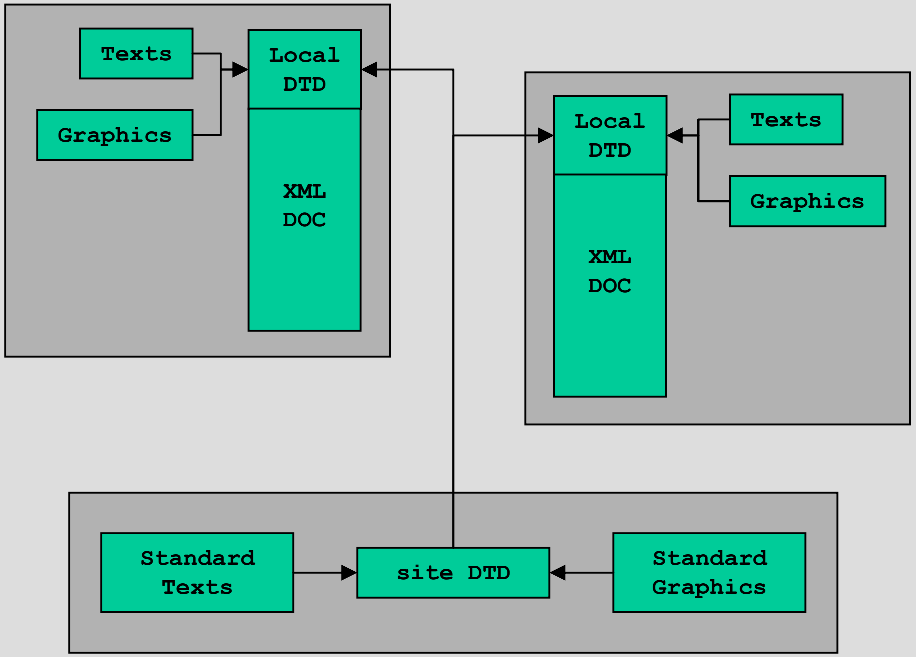
```
<?xml version="1.0" ?>
<!-- file: fred.xml -->
<!DOCTYPE FRED SYSTEM "fred-external-dtd.xml" [
<!ATTLIST GLOBAL
  BILL (a|b|c) "c">
]>
<FRED>
  <GLOBAL BILL="a">some content here…</GLOBAL>
</FRED>
```

```
<?xml version="1.0" ?>
<!-- file: fred-external-dtd.xml -->
<!ELEMENT FRED (GLOBAL)>
<!ELEMENT GLOBAL (#PCDATA)>
```

Hierarchy

- **Internal and External subsets form a hierarchical arrangement**
  - *allows modularisation and promotes reuse of (parts of) a DTD*
    - *the "pizza model"*
      - build a DTD based on a core and then add the toppings needed to create a particular document
      - e.g. Text Encoding Initiative (TEI) or Interactive Electronic Technical Manuals (EITM)

"There is a fixed order in which the external and internal DTD subsets are read and interpreted. First the internal DTD subset is read, and then the external DTD subset is read. If something is declared in the internal DTD subset, its declaration cannot be changed in the external DTD subset. However, some things (such as additional attributes) can be added to declarations."



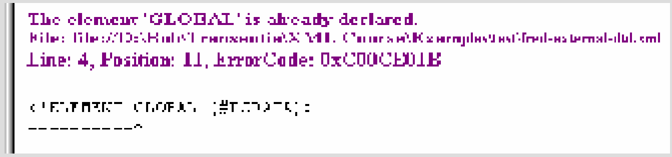
"Inheritance"

- **Hmmm...**
  - some authors seem to confuse reuse with inheritance
    - *in XML, modularisation of parts of a DTD can lead to reuse*
      - but is this inheritance?

```
<?xml version="1.0" ?>
<!ELEMENT NumberOfChapters (#PCDATA)>
<!ELEMENT CoverColour (#PCDATA)>
```

```
<?xml version="1.0" ?>
<!DOCTYPE TEXTBOOK SYSTEM "book-dtd.xml" [
<!ELEMENT TEXTBOOK (NumberOfChapters, CoverColour, Glossary)>
<!ELEMENT Glossary (#PCDATA)>
]>
<TEXTBOOK>
  <NumberOfChapters>1</NumberOfChapters>
  <CoverColour>Blue</CoverColour>
  <Glossary>Learning: A little knowledge is a dangerous thing!</Glossary>
</TEXTBOOK>
```

```
<?xml version="1.0" ?>
<!DOCTYPE COOKBOOK SYSTEM "book-dtd.xml" [
<!ELEMENT COOKBOOK (NumberOfChapters, CoverColour, Recipe)>
<!ELEMENT Recipe (#PCDATA)>
]>
<COOKBOOK>
  <NumberOfChapters>10</NumberOfChapters>
  <CoverColour>Red</CoverColour>
  <Recipe>Buy everything from supermarket</Recipe>
</COOKBOOK>
```



```
The element 'GLOBAL' is already declared.
File: file:///D:/Rob/Lecturenotes/XML Course/Examples/test/[illegible]-external-dtd.xml
Line: 4, Position: 11, ErrorCode: 0xC00CE01B

<!ELEMENT GLOBAL (#PCDATA)>
----------^
```

SYSTEM vs. PUBLIC

- External subsets referenced in one of two ways:
  - PUBLIC
    - *really a hangover from SGML*
    - *points to an entry in a* catalog *file*
      - entry in the catalog then supplies the URI

```
<?xml version="1.0" ?>
<!DOCTYPE FRED PUBLIC "-//IETF//DTD HTML//EN" [
  …
]>
<FRED>
  …
</FRED>
```

```
PUBLIC      "-//IETF//DTD HTML//EN"    "html.dtd"
```

      - entries should be registered, of course…
        - *string is actually (supposed to be!) meaningful*
  - SYSTEM
    - *subset referenced via a URI*
      - *Universal Resource Identifier*
        - *enhancement of URL*
        - *most often simply a URL*

```
<?xml version="1.0" ?>
<!DOCTYPE FRED
  SYSTEM "http://dtd.myorg.com/fred-external-dtd.xml" [
  …
]>
<FRED>
  …
</FRED>
```

Standalone

- May want to indicate that a document is self-contained and that there is no need to read an external DTD subset, even if one exists…

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

  – it is still an error if something that is not in the internal DTD subset is referenced…

Element Declarations

- **Structure Symbols**
  - the basis of the DTD ‘language’
  - similar to regular expression operators
    - *generally easy to grasp: similar to PERL language or UNIX shells or…*
  - ( )
    - *used for grouping*
  - ,
    - *specifies a sequence and associated ordering*
  - |
    - *separates two (or more) alternatives*
  - ?
    - *denotes an optional entry*
  - +
    - *required one or more times*
  - *
    - *required zero or more times*
  - *no symbol*
    - *must be as given*

```
<!ELEMENT EMAIL (TO+, FROM, CC*,
                 BCC*, SUBJECT?, BODY?)>
```

More Element Declarations…

- **Recall that an element is a container**

```
<!ELEMENT name content>
```

  - name provides a name for the container element
  - content may be:
    - *EMPTY*
      - element has no children
    - *ANY*
      - element can hold anything
    - *#PCDATA*
      - element can hold *parsed character data*
        - *textual data possibly containing entity references, etc…*
    - *content model*
      - as defined by the set of structure symbols used

Ambiguity/Mixed Content Models

- **Ambiguity is very undesirable in a content model**
  - consider:

```
<!ELEMENT confused ((this.one, that.one) |
                    (this.one, the.other.one))>
```

  - this is not considered an error, but it is a problem…
    - *loss of clarity in the model, greater memory requirements and processing time*
  - better rewritten as:

```
<!ELEMENT much.better (this.one, (that.one | the.other.one))>
```

"If you give XML software an element stream that can be processed in several different ways, it will normally select just one of those ways (probably not even telling you what it's done), and then continue processing. This situation can lead to confusion…"

- **Mixed content models allow for either highly-defined content or very 'free' content**
  - often very useful during development

```
<!ELEMENT mixed.content (#PCDATA | alpha | beta | gamma)*>
```

  - note *required* use of '*' in the definition

Attribute Declarations

- Slightly more complex
  - can define multiple attributes in one go
  - attributes may have a type; may have default values
  - attributes may also have other constraints placed upon them

```
<!ELEMENT BOOK (Title, Author*, Price)>
<!ATTLIST BOOK
  publisher CDATA #REQUIRED
  level (beginners | intermediate | advanced ) "beginners">
<!ATTLIST Price
  currency (A$ | US$) "A$">

…
```

```
<BOOK publisher="BigCo.">
  <Title>
    Gardening for Beginners
  </Title>
  <Author>
    G. Fingers
  </Author
  <Price>
    27.50
  </Price>
</BOOK>
<BOOK publisher="Kooks Publishers Ltd." level="advanced">
  <Title>
    Try Brain Surgery in the Comfort of Your Own Home
  </title>
  <Author>
    I. M. Knutts
  </Author>
  <Author>
    C. Rasé
  </Author>
  <Price currency="US$">
    4.45
  </Price>
</BOOK>
```

```
<!ELEMENT STUDENT (NAME)>
<!ATTLIST STUDENT STUDENT_ID ID #REQUIRED>
```

Attribute Types

- **Include:**
  - CDATA
    - *character data*
      - not parsed; can still contain entity references
  - ENTITY
    - *value must be an entity declared in the DTD*
  - ENTITIES
    - *whitespace delimited list of ENTITY values*
  - ID
    - *an identifier which the processor should ensure is unique* throughout the document
  - NOTATION
    - *value is a notation*
  - IDREF
    - *a reference to an ID elsewhere in the document*
  - IDREFS
    - *whitespace delimited list of IDREFs*
  - NMTOKEN
    - *attribute value is any mixture of “name token” characters (alphanumeric, ‘.’, ‘-’, ‘:’, ‘_’)*
  - NMTOKENS
    - *whitespace delimited list of NMTOKENs*
  - Enumerated
    - *attribute must* exactly *match one of the given values*

Ids & IDREFs

```
<?xml version="1.0"?>
<!DOCTYPE FAMILYTREE [
<!ELEMENT FAMILYTREE (PERSON*)>
<!ELEMENT PERSON (NAME, SPOUSE*)>
<!ATTLIST PERSON
          ID ID #REQUIRED
          FATHER IDREF #REQUIRED
          MOTHER IDREF #REQUIRED
>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT SPOUSE EMPTY>
<!ATTLIST SPOUSE
          IDREF IDREF #IMPLIED>
]>
<FAMILYTREE>
  <PERSON ID="p1" FATHER="p3" MOTHER="p4">
    <NAME>Fred Bloggs</NAME>
    <SPOUSE IDREF="p2"/>
  </PERSON>
  <PERSON ID="p2" FATHER="p5" MOTHER="p6">
    <NAME>Mary Sand</NAME>
    <SPOUSE IDREF="p1"/>
  </PERSON>
  <PERSON ID="p3" FATHER="p8" MOTHER="p22">
    <NAME>Charlie Bloggs</NAME>
    <SPOUSE IDREF="p4"/>
  </PERSON>
  <PERSON ID="p4" FATHER="p18" MOTHER="p46">
    <NAME>Amy Bloggs</NAME>
    <SPOUSE IDREF="p3"/>
  </PERSON>
…
```

Attribute Defaults

- **Include:**
  - #REQUIRED
    - *a value* must *be specified for the relevant attribute; a missing value is an error*
  - #IMPLIED
    - *the value is optional*
  - #FIXED
    - *this attribute must be given and must have the value specified in the declaration; anything else is an error*
  - default
    - *gives a default value for the attribute; if a value is not explicitly given for the attribute, the default value is assumed and passed to the processing application*

Optional Attribute

- Strictly speaking there is no such thing…
  - can use the following ‘trick’:

```
<!ATTLIST clever.element
  clever.attribute (A | B | C) "" >
```

  - allows:

```
<clever.element>
  something wonderful
</clever.element>
```

    - *the default value of "" would be assumed for clever.attribute…*

Conditional Inclusion

- **Facilities to turn parts of the *external* DTD subset on or off…**
  - IGNORE
    - *specifies that the enclosed portion of the DTD should be ignored by the processor*
  - INCLUDE
    - *the enclosed portion of the DTD should be made available to the processor*

```
<!-- file: fred-dtd.xml -->
<![IGNORE [
<!ELEMENT BCC (#PCDATA)>
<!ATTLIST BCC
  HIDDEN CDATA #FIXED "true">
]]>
<![INCLUDE [
<!ELEMENT SUBJECT (#PCDATA)>
]]>
```

```
<?xml version="1.0" ?>
<!DOCTYPE FRED SYSTEM "fred-dtd.xml" [
<!ELEMENT FRED (SUBJECT)>
]>
<FRED>
  <SUBJECT>blah blah blah…</SUBJECT>
</FRED>
```

  - useful with entities:

```
<!ENTITY % SECURE "IGNORE">
<!ENTITY % INSECURE "INCLUDE">
<![%SECURE; [anything goes here]]>
<![%INSECURE; [and any DTD definitions go here as well]]>
```

    - *note use of* parameter entities; *entities whose use is restricted to the DTD*

Parameter Entities

- Used to simplify and factor out commonality in declarations *in the DTD*

```
<!ENTITY % datemodel "(YEAR, MONTH, DAY)">
<!ELEMENT DOCUMENTDATE %datemodel;>
<!ELEMENT REVISIONDATE %datemodel;>
<!ELEMENT LASTACCESSDATE %datemodel;>
```

```
          <!ENTITY % source "CITATION | NAME">
          <!ELEMENT SENTENCE (#PCDATA | EMPHASIS | %source;)*>
          <!ELEMENT RESOURCE (%source;)>
          <!ELEMENT ORIGIN (%source; | UNKNOWN)>
```