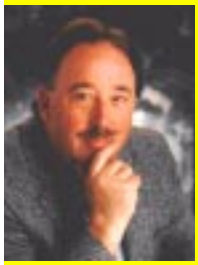


Jonathan Price

Making an XML DTD

Step by Step

An IDEA tutorial

Instructions:

You get step-by-step instructions on building each tag. No more abstract descriptions of syntax. You can see exactly how each tag is constructed, from beginning to end.

Diagrams:

You see the structure in a large diagram, so you can tell where each piece of punctuation goes, and what the components do.

Examples:

You get examples, so you can see exactly how the tags develop, and how they look when completed.

Answers:

If you have questions, we have answers. Addressing the most common questions that come up in the classroom, sidebars give you context and background on the tags.

The Communication Circle
918 La Senda, NW
Albuquerque, NM 87107
http://www.webwritingthatworks.com
ThePrices@theprices.com
http://www.theprices.com

Published in the United States of America.
First Publication: 2004
ISBN: 0-9719954-1-9

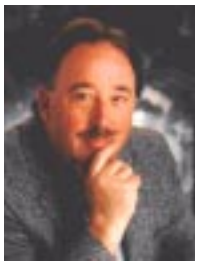
Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. The Communication Circle and Jonathan Price cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Contents



Jonathan Price

Making an XML DTD

Step by Step

Making a Valid XML Document

Validitating XML documents makes them more useful, consistent, and interchangeable.

An XML document **must** be well formed.

If not, you get a fatal error, and the parser refuses to complete its processing, so it passes nothing along to the browser.

An XML document **may** *also* be valid.

- A parser checks the structure of each document against the standard organization defined in the Document Type Definition (DTD).
- If you fail a test for validity, you just get an error.
- Validity constraints require declaring the document type, offering a document type definition, and following its rules.

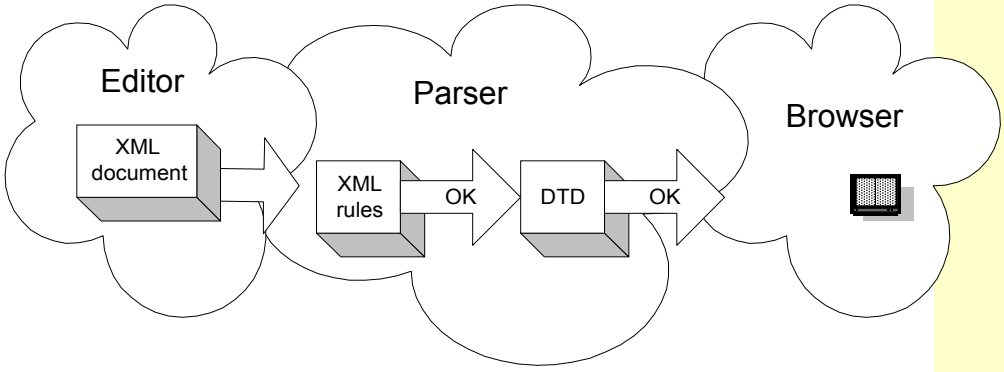
Validity guarantees that all documents referring to the same DTD have the same structure.

When a document is valid, the processing software can operate successfully on the contents.

The software knows what to expect, what tags to look for, in what order, what is included in what.

In effect, the standard organization turns the document into structured data.

- As a result, software can pick and choose content based on the user's choices, or preferences.
- Also, searching works the way it does in a relational or flat file database—faster, and more accurately than a fulltext search would.
- Pages are more likely to appear consistent in format, to the user.



1. In the XML declaration, inform the parser whether your definition of the ideal structure appears inside the document, or in another document.

Inside

If your document includes a full definition of the document type (DTD) inside the file, the document is a **standalone**. (Not recommended).

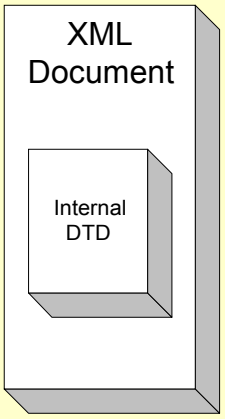
If the parser does not need any other document to understand this XML document, you announce that this one is a standalone, in your XML declaration.

Example of XML declaration

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

Example of a standalone document that can be validated because it includes its own DOCTYPE declaration, including an internal DTD:

```
<?xml version= "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE Copynotice [
    <!ELEMENT legalese ANY>
    <!ENTITY COPYDATE "2001">
    ]>
<Copynotice>
    <legalese>This book is copyright by ThePrices.com in the year &COPYDATE; and woe to anyone who tries to copy it electronically or in any way we cannot even imagine how.</legalese>
</Copynotice>
```



Standalone = "Yes"

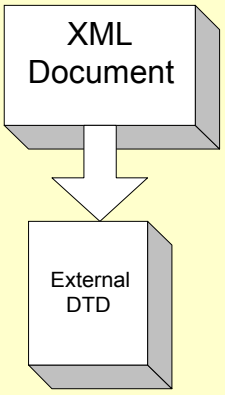
Outside, or Partially Outside

If your document refers to a separate Document Type Definition (DTD), your document is not a standalone. You must alert the parser that it will need to go find this other document (the DTD), following a line later in the file, called the DOCTYPE declaration.

Example of XML declaration that is not standalone.

```
<?xml version= "1.0" encoding="UTF-16" standalone="no"?>
```

You do not say where that file is, in the XML declaration. You just alert the parser that it should pay attention, because pretty soon you will get around to saying where that file is. The standalone attribute is a suggestion, not a command. Some parsers ignore it.



Standalone = "No"

2. Place a DOCTYPE declaration within the XML document's prolog.

You are telling the parser what kind of document this is, by identifying the Document Type Definition. Examples:

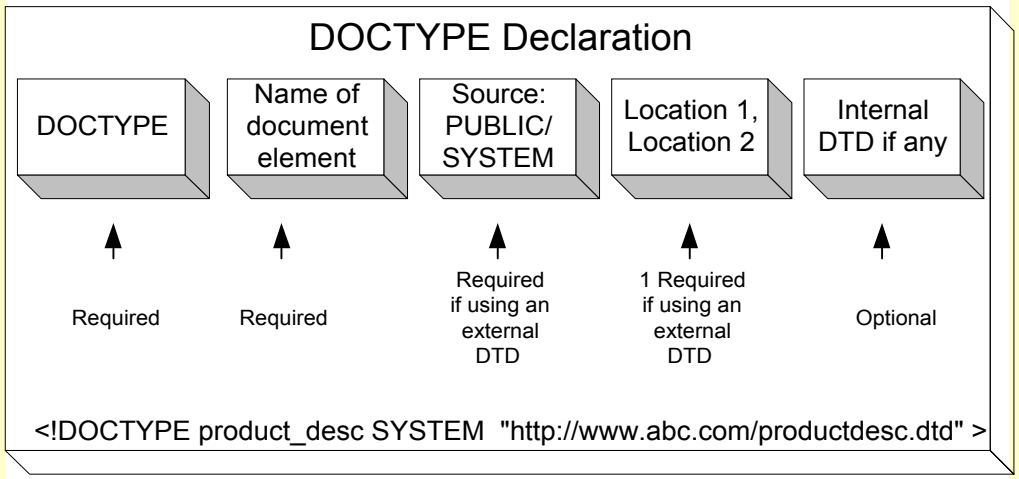
- Procedure
- Brochure
- DataSheet
- White.Paper

The DOCTYPE simply identifies the name of the type, and the location of the DTD that describes its standard structure. It may also include the DTD, though this is not a great idea, most of the time, or a subset of the DTD.

You may include one and **only one** DOCTYPE declaration per document.

If you want the document validated, you **must** include the DOCTYPE declaration. If you don"t care about validation, you don"t have to.

The parser reads the declaration and fetches the DTD (from wherever) and uses it to validate the document.



2a. Place the DOCTYPE declaration after the XML declaration, and before any elements.

2b. Start the declaration with XML tag delimiters, the angle bracket (<) and an exclamation point (!).

2c. Add the keyword DOCTYPE.

Your DTD has defined the logical structure and tags for a particular kind of document. You are about to name that document type.

Compare

The Document Type Definition (DTD) is a separate document that spells out the ideal structure and contents of that type of document.

The DOCTYPE declaration points to the DTD.

2d. Name the document element (the element that includes the whole content of the document).

The document element, also known as the root element, is the top element in the hierarchy defined in your DTD.

Example of the beginning of the DOCTYPE declaration

```
<!DOCTYPE catalog
```

Q: Is the document element the same as the root element?

A: Yes. Usually the name of the DTD is built on the root element's name. For instance, if you are making a catalog, you may name your DTD catalog.dtd, and the top element will probably be "Catalog." Logically enough, the name you put in the DOCTYPE declaration would be *catalog*.

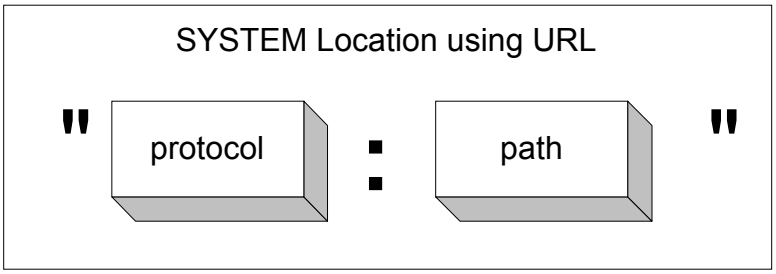
2e. Identify the type of source (SYSTEM or PUBLIC) and the location(s) of the DTD file.

If you can provide a specific location for the file, use SYSTEM. In the extremely rare case where you do not know where the DTD is, or it is so common that every parser knows it by heart, or can find it in an instant, you announce that the source is PUBLIC. Oddly, even in this case, you are encouraged to provide a second locator, which is, well, a path leading directly to the file on a particular system. Best practice: use SYSTEM.

SYSTEM

If you can provide a specific location, use SYSTEM, and give the Uniform Resource Identifier (consisting of a Uniform Resource Locator or Uniform Resource Name, i.e., URL or URN).

Using a URL



Example of SYSTEM locations using a URL

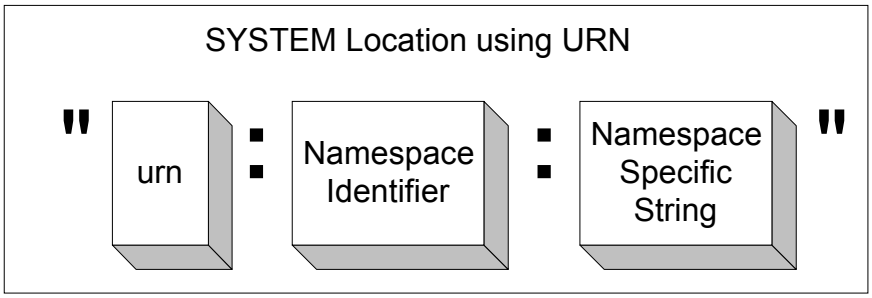
```
<!DOCTYPE FeatureList SYSTEM ""http://www.gobi.org/yurt.dtd"" >
<!DOCTYPE book SYSTEM  ""file:///usr/local/xml/docbook/3.1/
docbook.dtd"" >
```

Q: Why is a source SYSTEM?

A: The most common source is SYSTEM because, in theory, the file exists on your system, or maybe someone else's system.

Using a URN

The Uniform Resource Name is used occasionally to give a unique, location-independent name for a resource such as a standard DTD, or a person. To identify the context of the name—where it comes from—we must identify something called a namespace, a metaphysical cloud containing a bunch of specific names all related to a particular category of information (such as names for chemicals, or DVDs).

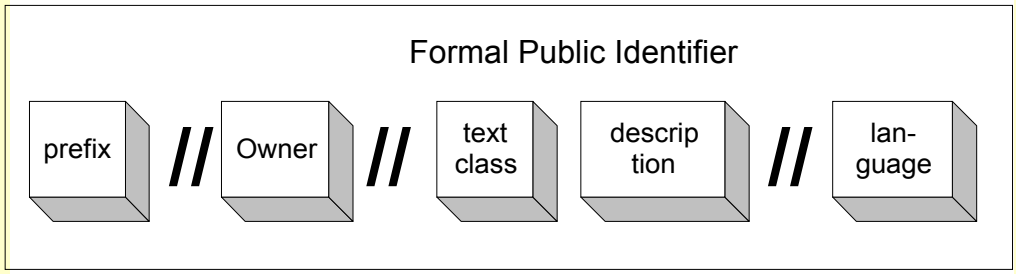


Example of a SYSTEM location using a URN

```
<!DOCTYPE FeatureList SYSTEM ""urn:w3.org:xhtml.dtd"" >
```

PUBLIC

If you are using a "well-known" industry-standard DTD, which may live on a local server, or be available over a private network, you can use a general, non-specific reference. (Very rarely used).



Example of PUBLIC location:

```
<!DOCTYPE book PUBLIC ""-//OASIS//DTD DocBook V3.1//EN"">
```

Because that location may be unavailable, or unknown, you should also give a second (SYSTEM) location, that is, a specific path to the file, without bothering to identify this as a SYSTEM location—within its own set of quotes.

Example of PUBLIC location followed by second location:

```
<!DOCTYPE FeatureList PUBLIC ""ManchuCollaborativeYurt""
""http://www.gobi.org/yurt.dtd""
```

2f. If you are including all of the DTD within the document, put that here within square brackets, instead of the source and location.

For full details on the DOCTYPE declaration, please see *An ABC of XML Tags*.

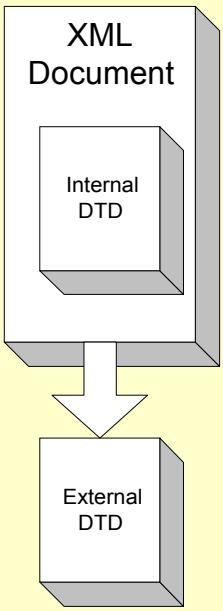
Example of an internal DTD within the DOCTYPE declaration, defining a document element called ALERT, which can contain any kind of content.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE ALERT
        [<!ELEMENT ALERTANY>]>
<ALERT>Emergency: Condition Yellow.</ALERT>
```

2g. If you are appending an internal subset, put it after the external location.

Your declaration can point in two directions—outward, to an external DTD file, and inward, to some local additions or refinements. You still have one DOCTYPE declaration. It just has two parts—the **external** subset and the **internal** subset.

Key: The **internal** subset takes precedence over the **external** subset.



Standalone ="No"

Example of a DOCTYPE declaration with external and internal subsets.

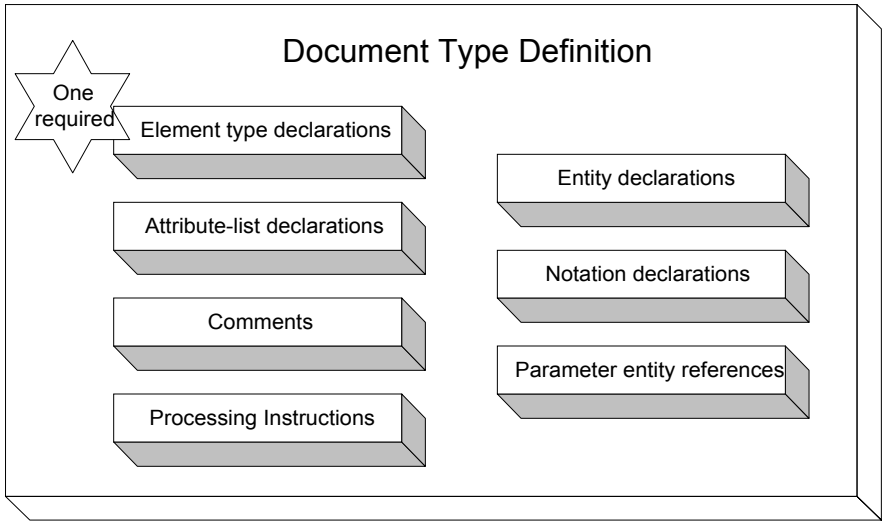
```
<?xml version="1.0"?>
<!DOCTYPE BIO SYSTEM ""file:///site/bio.dtd"" [
<!ENTITY ROUSSEAU SYSTEM "" file:///site/rousseauphoto.jpg""
NDATA JPG>]>
<BIO>
<TITLE>Rousseau the Philosopher</TITLE>
<DATE>June 28, 1712</DATE>
<SUMMARY>Born on this day, Jean-Jacques Rousseau became a
philosopher, writer, and musician. He wrote The Social Contract,
which provided ideas to the American Revolutionaries, and Confes-
sions, which deepened the genre of memoirs.
&ROUSSEAU;
</SUMMARY>
</BIO>
```

Rousseau the Philosopher
June 28, 1712

Born on this day, Jean-Jacques Rousseau became a philosopher, writer, and musician. He wrote **The Social Contract**, *which provided ideas to the American Revolutionaries, and* **The Confessions**, *which led to thousands of romantic memoirs.*

2h. End with a greater-than character.

3. Lay out your document's standard structure in a Document Type Definition (DTD).



- **Element type declarations**—The objects a document may contain, arranged in a single hierarchy.
- **Attribute-list declarations**—The names of the characteristics you can use to describe a particular element, along with a description of the attributes" data types and default values.
- **Comments**—Notes to yourself or your team.
- **Processing instructions**—Commands to be passed to the application that will be manipulating the document after the parser gets through with it.
- **Entity declarations**—Boilerplate chunks of text, special characters, macros, repetitive content from external sources
- **Notation declarations**—Non-XML content coming from outside the document, such as images, their data formats, and the programs that will be used to provide the information
- **Parameter entity references**—A collection of any of the other items here, to be inserted with a single mention.

The DTD is **usually** a separate document, but it can appear as an appendage to the DOCTYPE declaration.

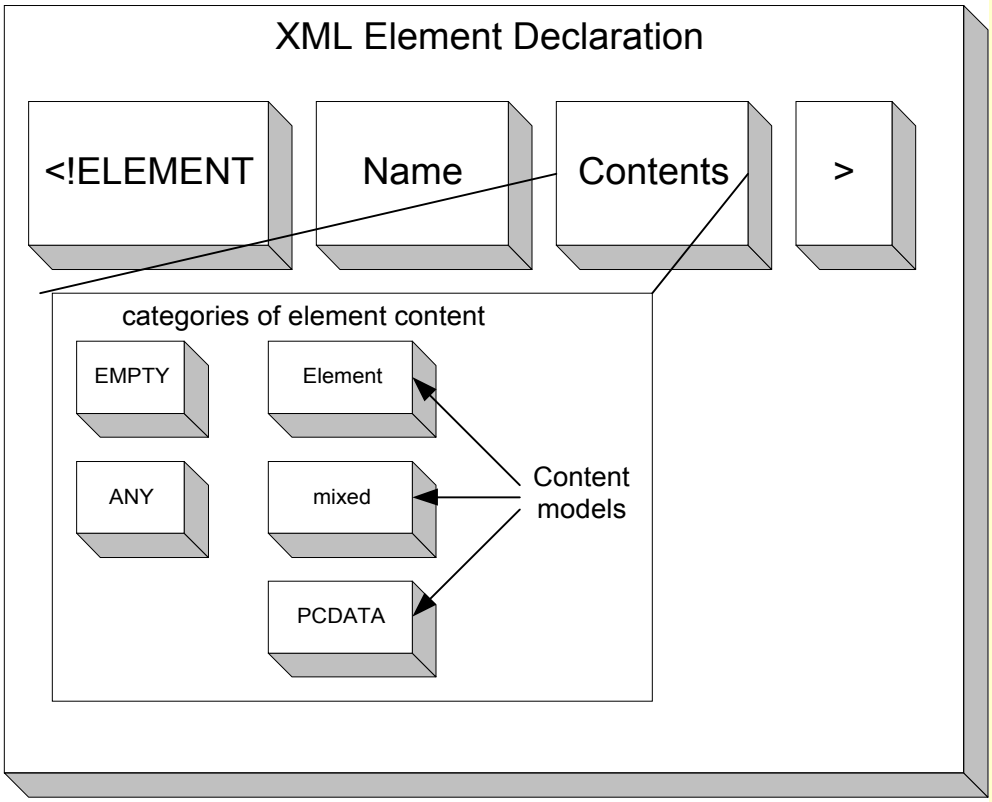
All statements in the declaration begin with <! and end in >.

Keywords such as ELEMENT and ATTLIST (list of attributes) appear in ALL CAPS.

4. Declare elements.

Every document must have at least one element—the top level, or document element. Remember: that's all that you need to produce a well-formed XML document. Elements are pieces of content.

Within that document element, which we are declaring in the DTD, there may be child elements, character data (text), or nothing at all, in various combinations.



4a. Begin with <!

4b. Say what you are defining IN ALL CAPS (an ELEMENT).

This is a KEYWORD (predefined in the XML standard).

4c. Name the document type you are defining (a generic identifier, or GI).

Examples: procedure, chapter, FAQ, invoice, white.paper, thank_you, registration_form.

Q: Why are we talking about a document type?

A: Type means *category*, or *class*. You are defining a class of document. When you actually create a document with real content, that particular document is viewed as an instance of the class.

4d. Specify the type of content in the element.

There are five categories of element content:

- **EMPTY**—This is just a **category**. It has no content. This element does not contain any text or child elements. You can attach attributes to this empty element, though.
- **ANY**—This is just a **category**. Since it can include anything, it, too, lacks any content model. You can include any well-formed XML data, such as text or elements.
- **element**—Requires that you define a **content model**. An element contains only elements. No extra text is allowed in this type of element. You put the names of the child elements inside parentheses, when making the structure explicit.
- **mixed**—Requires an extremely loose **content model**. May contain ordinary text and/or elements. You put the names of the elements and the text inside parentheses, separated by pipes, and add an asterisk after the closing parenthesis.
- **PCDATA**—A category for text (characters), usually. May include entity references such as those to oddball characters. You put the keyword #PCDATA inside parentheses.

- If you want to point to another file such as a graphic, but have no other content, put EMPTY in, without any parentheses around it.

```
<!ELEMENT breakpoint EMPTY>
```

Note that the keyword EMPTY appears in all capital letters, without parentheses around it.

- If you allow any darn thing, without any structure, as content, use ANY.

Of course, using ANY misses much of the point of XML. This category of content can contain any well-formed XML data, so there is no particular validation within such an element, and we know nothing of the organization of the chunks of content within this element.

Note that the keyword ANY appears without parentheses, in the declaration.

Example

We expect articles to come in from many different sources, and we have no control over their content, tagging, or formatting. To accommodate these, we define the content as, well, ANY.

```
<!ELEMENT article ANY >
```

Q: Why would you want an **empty** element?

A: If you want to mark a spot in the document, for the sake of later formatting, or if you want to take advantage of the attributes of the element without having any content visible to the user.

Even though the element has no official content, its attribute may point to an external file, or give some information that you do not want users to see. (More about attributes later).

- If the element should contain only regular character data, put #PCDATA in parentheses.

In the document itself, you can insert a character reference or a predefined entity reference so as to get special symbols or boilerplate into the character stream.

You cannot insert elements into the character stream, though. Saying that the content will be PCDATA means that the element contains no child elements. If you try to sneak one in, you get a validity error.

You may put a space character between the parenthesis and the #PCDATA keyword, for legibility.

Caution: PC Data may not be politically correct. Its characters have been parsed. That is, the parser looks inside this text for any XML markup. Therefore, you cannot drop in

- Right angle brackets >
- Ampersands &
- 2 square brackets followed by a right angle bracket]]>

Why? Because the parser will figure you are issuing some kind of markup. (Instead, use character reference or predefined entity references).

Example: Our title should be text, and we do not want it to contain any child elements.

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

- If the element includes one or more other elements, name them within parentheses.

These are the children of the element. (You are saying that the element contains other elements, but, outside of those, cannot contain random bits of text, known as #PCDATA.)

You now need to define the content model, that is, the **elements** that are to be included, the **order** of those elements, and the **number** that are required or optional, as explained in **the next few items**.

Example

```
<! ELEMENT author (first_name, last_name)>
```

- If the elements must appear in a certain order, put them in that order, separating them with commas.

Example

In the book element, we want to see the title first, then the author, ISBN, and binding, in that order. We want all of these, and we want them in this order.

```
<! ELEMENT book (title, author, ISBN, binding)>
```

- If you want to allow only zero or one instances of the child element, append a questionmark.

Example

In the front matter element, we want to have only one of each child element—or none. We do not want to have more than one preface, acknowledgements section, or introduction. And we can live with none. We want to make sure that the introduction, if any, follows the acknowledgements section, if any, which comes after the preface, if any.

```
<!ELEMENT front_matter (preface? acknowledgements? intro?)>
```

- If you want to allow zero or more instances of the child element, append an asterisk.

Example

In our academic journal, we have learned that many authors have several different subtitles, where other authors have none. To accommodate the professors, we tell the parser to accept the title element as long as it has a main_title element, and to accept any number of subtitles, or none.

```
<!ELEMENT title (main_title, subtitle*)>
```

- If you want to insist that there be 1 or more instances of the child element, add a plus sign.

Example

In our procedure element, we insist that there be a procedure_name, and we can live with no introduction, or one (but no more). We insist that there be at least one step, but we accept multiple steps.

```
<!ELEMENT procedure (procedure_name, introduction?, step+)>
```

- If you want to include one element out of a list of choices, use a pipe to separate them.

This is an exclusive *or*.

Example

In the blurb element, you can have one review comment, **or** one vendor description, **or** one feature list. You must have one of these.

```
<!ELEMENT blurb (review_comment | vendor_description |
feature_list)>
```

- If you want to include parsed character data plus zero or more child elements, in any order, and with any number of occurrences (zero or more), so that all the elements are, in fact, optional, use the **mixed** content model, adding the

elements like this:

1. Open parentheses
2. Start with #PCDATA.
3. List elements separated by pipes.
4. Close the parentheses.
5. Add an asterisk outside the closing parenthesis, to indicate that the whole shebang can appear zero or more times.

Example

In the new blurb, we accept the possibility that there will be absolutely nothing. On the other hand, we also accept any number of pieces of text, review-comments, vendor_descriptions, and feature-lists, in any order. We are easy.

```
<!ELEMENT blurb (#PCDATA | review_comment |
vendor_description | feature_list)*>
```

Summary of operators

Order

- Comma (,). The comma means one element must follow the next, in this order. a, b, c.
- Pipe (|). The pipe indicates choice. You must include one of the items.

Number

- Questionmark (?). Zero or one must appear. That is, the item is optional, but if it appears, it can only appear once.
- Asterisk (*). Zero or more instances are acceptable. It is optional, and may appear several times, if you want.
- Plus sign (+). One or more should appear.

4e. End with a closing bracket >

Example: Complete element declaration.

```
<!ELEMENT Faq.menu (menu.title, heading+)>
```

Example of complete DTD and document

The parser compares the DTD with the document to determine if the document is valid.

DTD:

```
<!ELEMENT cd (title, artist+, label, price, cd_description)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT cd_description (#PCDATA)>
```

VALID XML ELEMENT IN THE DOCUMENT:

```
<cd>
    <title> Hurricano </title>
    <artist> Los Nuevos </artist>
    <label> La Senda </label>
    <price> $14.95 </price>
    <cd_description> A wonderful symphony of New Mexican
lyrics, from a leading punk fusion heavy metal group.</
cd_description>
</cd>
```

Challenge 0:

Create a Memo element with child elements for To, CC, BCC, From, Date, Re, and Body.

Challenge 1:

Create a Report element that contains child elements for author, title, category, abstract, keywords, heading1, heading2, and paragraph.

Challenge 2:

Is this element valid, using the DTD for cd?

```
<cd>
    <title> Hurricano </title>
    <artist> Los Nuevos </artist>
    <price> $14.95 </price>
<label> La Senda </label>
</cd>
```

Q: What's the answer?

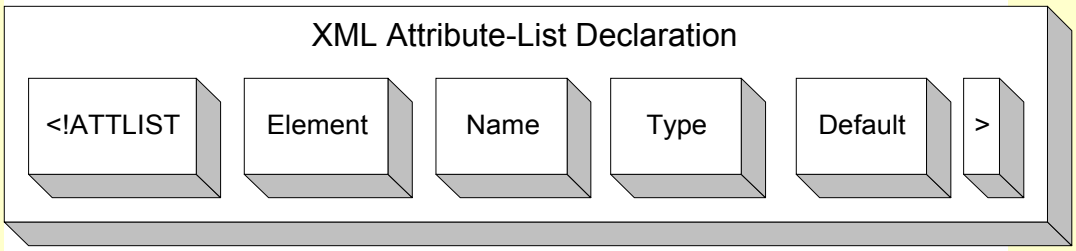
A: Take a look for answers to the challenges, beginning on page 54. Remember that in some cases, you may have a perfectly good answer, even if it is different from mine.

Challenge 3:

Create a diagram showing what elements form part of the book.intro element, and how they are arranged, hierarchically. Imagine a document that has at least one of each element.

```
<!ELEMENT book.intro (H1, book.overview, where.am.I?,
whats.new?, general.cautions?)>

<!ELEMENT H1 (#PCDATA)>

<!ELEMENT book.overview (p+, chapter.title+)>

<!ELEMENT chapter.title (#PCDATA)>

<!ELEMENT where.am.I (p+, site.map)>

<!ELEMENT site.map (area.title+)>

<!ELEMENT area.title (p?, button)>

<!ELEMENT button (#PCDATA)>

<!ELEMENT whats.new (p+, button+)>

<!ELEMENT general.cautions (p+, button+)>

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

5. Declare the attributes of an element.



An attribute describes an element.

An attribute is an adjective to the element's noun.

- An attribute may provide information about the information (date of issue, author, subject).
- An attribute may be thought of as a property of the element. If the element is a barn, the attribute called color probably has the value of red.
- An attribute may also be thought of as a field with various possible values.
- You can declare lots of attributes for one element—a list of atrributes.

Rule of thumb

If the user must read the information, it is probably **an element**. If you or software are the intended reader, you are looking at an attribute.

Not an attribute

- No, an element cannot be turned into an attribute for another element. (If you are tempted, rethink the relationship between the two elements).
- **Parts of an element** should probably be made into child elements, not attributes. In a book, chapters are child elements to the book as a whole. Attributes for the book might be the number of copies that have already been sold, the inhouse editor, the author's email.

Constraints

- Any one attribute can appear **only once per element** (as opposed to elements, which can be defined as appearing several times).
- Attributes contain **only text** (without XML structure) or lists of text.
- Attributes **cannot contain elements**, and you cannot have a sub-attribute of some kind.
- In the document, the value of an attribute appears inside single or double quote pairs. Example: Description="Good"

Q: If I define attributes in a certain order in my DTD, do I have to use them in that order in the document?

A: No. Attributes may appear **in any order** in the document. If your DTD specifies them in the othe order A, B, C, they may appear in any sequence in the document. C, A, B is perfectly acceptable.

Q: I want to have the same information about an element appear several times. Can I use an attribute?

A: No. You can only use the attribute once for each instance of the element. Workaround: Create a child element, and insert it into the parent element with an operator indicating how many instances you will allow, or create an overarching element that includes your original element plus one or more instances of the new element.

5a. Begin with <!

```
<!
```

5b. Say what you are defining (an attribute list). Keyword: ATTLIST.

```
<!ATTLIST
```

5c. Name the element whose attribute (or list) this is.

```
<!ATTLIST book
```

5d. Name the attribute (a characteristic of the element).

```
<!ATTLIST book covertype
```

5e. Describe the type of value you can assign to this attribute in the document.

```
<!ATTLIST book covertype CDATA
```

Attribute types enforce constraints on the values entered in the document.

The constraints of a type apply to the value AFTER it has been **normalized**.

To normalize the value entered in the document, a parser turns the incoming text (markup and data) into data by carrying out these steps:

1. Strips out the **surrounding quotes**.
2. Replaces any **character references** with the characters referred to. For instance, & is replaced by the ampersand itself.
3. Replaces **general entity references**. If you define your initials as a general entity reference standing for your full name, the parser replaces your initials with your full name.
4. Replace any **new line characters** (paragraph return, line feed) with space characters.
5. If the attribute type is something called tokenized, removes any **leading or trailing spaces**, and reduces multiple spaces between tokens to single spaces.

What type of value to use

- For simple text (character data), use CDATA (character data). You do need to use character references for oddball characters such as the ampersand, and you can't use the same kind of quotes you are using to surround the value.
- For one of a series of values explicitly defined in the DTD, use Enumeration, or Enumerated Values (a choice list with OK values).

- For a unique identifier for this element instance, use ID (Identifier). The identifier must be a text string that follows XML rules for names. The first character may only be a letter, an underscore, or a colon.
- For a reference to an element that has an ID attribute with the same value as you put hee, use IDREF (Identifier reference).
- For a list of these references, separated by white spaces, use IDREFS (an Identifier references list).
- For a name token, use NMTOKEN. That is, a piece of text that conforms to the XML name rules, except that the first character can be any character you could put in a name, anywhere.
- For a list of name tokens separated by white spaces, use NMTOKENS.
- For the name of a pre-defined entity, use ENTITY (a strange form from outer space).
- For a list of predefined entities, separated by white space, use ENTITIES.
- For a notation type explicitly defined elsewhere in the DTD, use NOTATION. Example: US Date, ISO date, jpg. You can offer a series of choices separated by pipes.

Challenge 4:

Which type would you choose for the following attributes? (Some attributes may have several possible types).

- One of three sizes. (No other sizes allowed).
- A Social Security Number.
- The name of an image file, previously defined as an entity within your DTD.
- An indication of file type, chosen from several already defined as notations.
- A unique non-numeric password.
- One or more elements, which you want to refer to.

Challenge 5:

Which type would you choose for the following attributes? (Some attributes may have several possible types).

- One of three kinds of book cover. (Hard cover, library, or soft cover).
- A serial number for each product.
- A pointer to a product, using its serial number.
- A code that starts with an underscore.
- The name of an entity that describes an external file.
- The date format being used.

- If the value will be just regular text, known gloriously as character data (a string of text), use the CDATA type. This is the easiest, loosest type.

Any characters recognized by your character set, such as Unicode or Ascii, except the XML restricted characters—the ampersand, angle brackets, apostrophe, and double quote marks. For these, use the five built-in entities for less than, greater than, ampersand, apostrophe and quotes— < > & ' & quot; no other markup, please!

Example declaring the "bundle" attribute for the element called product, which can be any text. The attribute is optional, or implied.

```
<!ATTLIST product bundle CDATA  #IMPLIED>
```

What that would look like in a document:

```
<product bundle="Extreme">
Aerofrizzee
</product>
```

Challenge 6:

Create an element called PRODUCT, and add an attribute called DESCRIPTION, which will contain several sentences discussing the features and benefits of the product. Also, an attribute called Price.

Challenge 7:

Create an element called Partner, and add attributes for MISSION, which is usually a lengthy paragraph, and HEADQUARTERS_CITY.

- If you want to offer a list of choices, use an enumeration (a list of acceptable values separated by pipes)

Each value must follow the rules for a legal XML name token. That is, it consists of one or more letters, digits, periods, hyphens, or underscores. It can contain one colon, but not in the first position. No spaces, ampersands, angle brackets. None of the quotes used to surround the value.

You can have as many values as you want.

Users **must** enter a value. Empty is not a valid value.

Tip: You can use these NameTokens for numbers, using the hyphen as a minus sign, the period as a decimal point, and special characters for fractions.

If your later processing cannot handle space characters within a value, or other forms of punctuation, use Name Token values so that you can verify the value does not contain any debris like that.

Outtake from the DTD

```
<!ELEMENT Employee (FirstName, MiddleName, LastName) >
<!ATTLIST Employee
    honorific (Mr. | Ms. | Dr. | Rev.) # IMPLIED
    suffix (Jr. | Sr. | III) #IMPLIED>
```

What that would look like in a document:

```
<Employee honorific="Rev." suffix="III">
<FirstName>George</FirstName>
<MiddleName>Vidal</MiddleName>
<LastName>Blair</LastName>
</Employee>
```

Challenge 8:

Create an element called employee, and an attribute called STATUS, which will allow only the values fulltime, parttime, and contractor.

Challenge 9:

Create an element AGENT with the attribute SECURITY, which can take only the values None, Secret, Top_Secret, and Eyes_Only.

- If you want the value to be a unique identifier, use ID.

The value for this attribute must be **unique**, for each element in a document.

(No element may have more than one ID attribute, and the value for each element's ID attribute is unique, within the document).

Often you turn to software to generate these identifiers, but you might insist on clerks entering, say, a person's badge ID, a Draft code, or a Student ID--if these begin with a letter.

The ID lets you refer to a **specific part** of the document. You can use it as the **target of a link** (using IDREF), or to set up a one-to-one relationship between two elements, in different documents, to be used by a database. To a relational database, the ID is the primary key for the row.

Caution: to save your sanity, do not use the letters *ID* as part of any other kind of attribute's name.

The value must begin with a letter or underscore, followed by zero or more letters, digits, periods, hyphens or underscores. You can have one colon, but not in the first position. **Note**: If you are drawing IDs from a relational database's primary keys, make sure that those values meet the XML standards.

The ID attribute must have a default of #REQUIRED or #IMPLIED. It cannot be #FIXED (because then it wouldn"t be unique when you created a second instance of the element in the same document), and there can be no

default value (for the same reason). If you intend to link to the element, its ID value is REQUIRED.

```
<!ATTLIST person badge ID #REQUIRED>
What that would look like in a document:
<person badge="ss307-55-6675">
Herbert Walker
</person>
```

Challenge 10:

Create an element COURSE with an attribute for a course identifier, called the code.. Each course code is unique, and begins with a 3-letter code for the department. For instance, an English class in Shakespeare, English 400, would have the code Eng400.

Challenge 11:

Create an element CONTRACT with an attribute being the internal contract number, which beigns with the text "CT."

- If you want the value to be a reference to an element that has an ID attributre with the same value, use IDREF. For several, use IDREFS.

The only valid entry for an IDREF will be an ID that exists somewhere within the same document.

If you are depending on software to manipulate the IDREFs, make them required, because if they are left out, the parser looks to any defaults it can find at the document level.

An ID is the target of a link. You can have multiple IDREFs all pointing to the same element's ID.

If you have a list of IDREFS, separate them with white spaces.

Example of IDREF attribute

In the author element, we want to be able to link to a bio, which we expect to be somewhere else in the same document.

```
<!ATTLIST author bio IDREF #REQUIRED>
```

What that would look like in a document:

```
<author bio="auth10">
William Shakespeare
</author>
```

Example of IDREFS attribute

```
<!ATTLIST product
Recommend IDREFS #REQUIRED
SKU ID #REQUIRED
>
```

What that would look like in a document, pointing to the IDs B8997, B8976, and A5467.

These multiple values, separated by spaces, identify targets of potential links.

```
<product SKU="A6789" Recommend="B8997 B8976 A5467">
Whirlybirdarama
</product>
```

Challenge 12:

Create an element called HOWTO, which may have a number of TIP elements referenced. Also create a TIP element. Each TIP has a unique TIP_CODE.

Challenge 13:

Create an element called SCENT, with an attribute that points to the IDs of all the products that contain that scent.

- If the value will be limited to characters that are valid for XML names (letters, digits, period, dash, underscore, colon, alternate quotes)… use NMTOKEN. For several, use NMTOKENS.

The word *name* was already taken, so for this kind of value, the XML squad came up with the term *Name Token*.

You can't put a colon first, but you can put any other valid name character first: letters, digits, periods, hyphens or underscores. You can have one colon. Remember: the white space between items in the plural version, NMTOKENS, gets stripped out by the parser.

NMTOKENs are more liberal than other XML names, but impose some validation, compared to CDATA, which is wild and crazy. Compare:

- **Most restricted**: An enumerated list, because you can only use those specific values.
- **Moderately restricted**: Name Tokens, which follow the rules for XML names (except that any name character can appear at the start of the name token), while allowing you to make up any content you want.
- **Loose**: CDATA, which has no limits on characters, or content (except that even here you cannot use the ampersand, angle brackets, and so on.)

Caution: a parser does not separate the various Name Tokens in a list, so the receiving application (the browser, for instance) has to make sense of them.

Rule of thumb: Most people avoid Name Tokens, going for either CDATA or enumerated lists.

Example of a Name Token being used for values beginning with a number.

```
<!ELEMENT BOOK (#PCDATA)>
    <!ATTLIST ISBN NMTOKEN #REQUIRED>
```

That would show up in the document like this:

```
<BOOK ISBN="8-66768-999-2">The Best Thing on TV</BOOK>
```

Example of NMTOKENS set up to accept a series of values.

```
<!ELEMENT SUSPECT (#PCDATA)>
<!ATTLIST SUSPECT AKA NMTOKENS #IMPLIED>
```

That would show up in the document like this:

```
<SUSPECT AKA="Moe Tiny Razor">Johnny Moses</SUSPECT>
```

Challenge 14:

Create an element called PART and a required attribute to contain the date that part was created, in the format 2001-06-31.

Challenge 15:

Create an element called CONFIRMATION with a required attribute for the unique number of the message, which comes in the form _000909.

Challenge 16:

Identify the likely reasoning behind the attribute types in the following declarations:

```
<!ELEMENT INVENTORY (ITEM*)>
<!ELEMENT ITEM (#PCDATA)>
<!ATTLIST ITEM
    Code ID #REQUIRED
    Description CDATA #IMPLIED
    Price NMTOKEN #REQUIRED
    Suggestion IDREFS #IMPLIED
    Featurelist IDREF #REQUIRED>
```

Discover if any parts of the following outtake from a document using this DTD are invalid:

```
<INVENTORY>
<ITEM Code="a21a00" Price="14.55" Suggestion="a31a01 a42b21">
Palm VII Windscreen </ITEM>
<ITEM Code="a21a00" Price="79.95" Description="A tornado in
your hand, a blizzard blasting out of four sturdy blades, your personal
fan. Just the thing for your trip to the beach. Don"t bother to go in
the water. Just turn on your Buzzer." Suggestion="a55c66 a65a43"
Featurelist="FL81">The Buzzer</ITEM>
</INVENTORY>
```

- To refer to an external file (known to XML as an unparsed entity) use the ENTITY type. For a list of several unparsed entities, separated by white space, use ENTITIES.

If you need to link to, refer to, or include objects outside this document, such as images, CGI scripts, text files—stuff that should not be fed through the parser—you can use the attribute type ENTITY. Its full name is: unparsed entity.

The value must be a legal XML name.

To use an ENTITY type, you have to take these actions:

1. **Declare a notation (for instance, png).**

```
<!NOTATION png SYSTEM "http://www.theprices.com/apps/
PNG_Viewer.exe">
```

2. **Declare an entity (showing the path to the other file). Example: Buddy.**

```
<!ENTITY Buddy SYSTEM "http://www.theprices.com/Images/
Buddy.png" NDATA png >
```

3. **Declare an element, with the attribute of the type ENTITY.**

```
<!ELEMENT employee (#PCDATA)>
  <!ATTLIST employee photo ENTITY #IMPLIED>
```

4. **Use the entity as a value when you create an instance of the element.**

   In the document, that might appear as:

```
<employee photo="&Buddy;">Buddy Llorona</employee>
```

Note: For ENTITIES, you can include several entities within the quotation marks, separated by spaces.

Example of ENTITY attribute type

Here the notation and entity called panorama have already been declared in the DTD:

```
<!ELEMENT image EMPTY>
  <!ATTLIST image source ENTITY #REQUIRED>
```

In the document the link to the entity called **panorama** might look like this:

```
<image source="panorama" />
```

Example of ENTITIES attribute type

Here the notation and entities called cityscape, bridge, and Eiffel have already been declared:

```
<!ELEMENT image EMPTY>
    <!ATTLIST image source ENTITIES #REQUIRED>
```

In the document the links to 3 of these entities:

```
<image source="cityscape bridge Eiffel" />
```

Challenge 17:

Assume that elsewhere in your DTD you have declared that png, gif, and jpg are valid notations, and that you have declared the following pictures as entities: OurLogo, OurLogosmall, OurLogoBW.

Declare an element called LOGO, which is empty, but has as an attribute a pointer to one of these entities. Then write the part of the document that invokes the small logo.

Challenge 18:

Assume that elsewhere in your DTD you have declared that PDF, htm, and html are valid notations, and that you have declared the following files as entities: WhitePaper001, WhitePaper002, and WhitePaper003. Declare an element called WHITEPAPER, and an attribute pointing to an entity. Write the part of the document that points toWhitePaper001.

- If your browser or later applications need to know what kind of files these external unparsed entities are, use the NOTATION type.

This allows you to offer a choice list of possible notations, with pipes between them. The parser will make sure that you have declared the notation type in the DTD. (We"ll talk about declaring a notation later).

Example

```
<!ELEMENT picture EMPTY>
<!ATTLIST picture
     src CDATA #REQUIRED
      type NOTATION (png | gif | jpg) #IMPLIED
>
```

Document

```
<picture src="http://www.theprices.com/Images/noah.jpg"
type="jpg">
```

Challenge 19:

Assume that you have already declared that you recognize png, gif, and jpg as notations. You have also declared an ENTITY called JPRICE, which points to Jon.jpg. Create an element called Face, with attributes for a Social Security Number, a student number, an image source, and its file format. Then make up a part of a document with my face.

Challenge 20:

Assume that you have already declared that you recognize bmp, tif, png, gif, and jpg as notations for filetypes. You have also declared an entity named GUERNICA pointing to a bitmap of the painting. Create an element called Painting, which has no text, but has attributes for a unique accession number, a unique inventory number, an image source, and its file format. Then make up a part of a document for Picasso's *Guernica*.

5f. Say whether the attribute content is required. (Default declaration).

- If the content is required, put #REQUIRED

Example

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

<!ATTLIST citation reference ENTITY #REQUIRED>

<!ENTITY Newsweek SYSTEM "http://www.newsweek.com/
index.htm" NDATA htm>
```

In the document:

```
<citation reference="Newsweek">The Internet now accounts for
8% of all retail sales.</citation>
```

- If the attribute must always have a particular value, put #FIXED and the value in quotes.

Example

To tell visiting robots that a heading is a title element it is looking for, you might write:

```
<!ELEMENT heading #PCDATA>

<!ATTLIST heading title-element CDATA #FIXED "TITLE-ELE-
MENT">
```

If your attribute is a list of enumerated values, but one of them is the default, you can put that in quotes, after the parentheses:

<!ATTLIST VOTE CHOICE (yes | no) "yes" >

That implies that the default is required.

- If a value is not required, but optional, and may or may not appear, put #IMPLIED

Use if the value is not always available. Make sure that your users will not mind if the value is missing, or will have a good idea what that means.

Example

To insert the size of a file, if known, you could write:

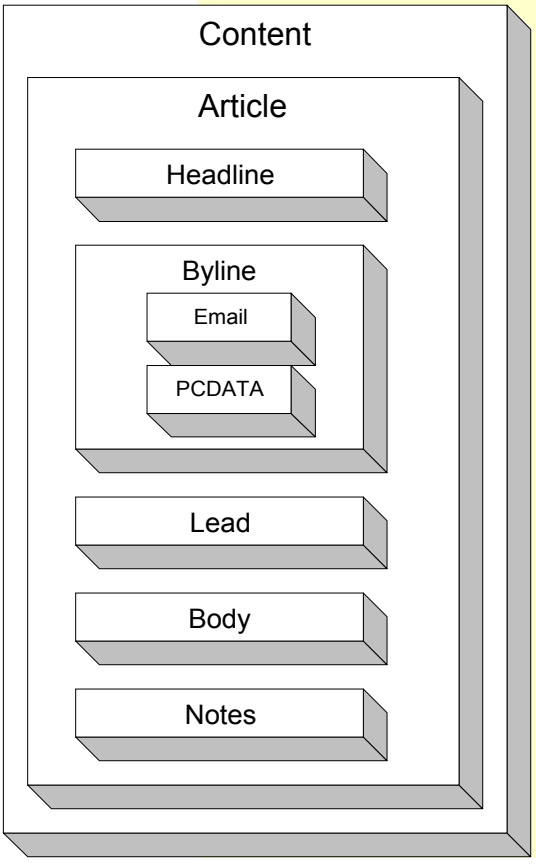
```
<!ELEMENT file #PCDATA>
<!ATTLIST file size CDATA #IMPLIED>
```

5g. End with a closing bracket >

```
<!ELEMENT Faq.menu (menu.title, heading+)>
    <!ATTLIST Faq.menu anchor ID #REQUIRED>
```

Use attributes to identify elements that might be targets for links, or anchors (hot spots).

```
<!ELEMENT CONTENT (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE, BYLINE, LEAD, BODY, NOTES)>
<!ATTLIST ARTICLE
AUTHOR CDATA #REQUIRED
EDITOR CDATA #IMPLIED
DATE CDATA #IMPLIED
EDITION CDATA #IMPLIED
ARTICLE_CODE ID #REQUIRED>
<!ELEMENT HEADLINE (#PCDATA)>
<!ATTLIST HEADLINE TARGET ID #REQUIRED>
<!ELEMENT BYLINE (#PCDATA | EMAIL)*>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ATTLIST EMAIL biolink IDREF #REQUIRED>
```



Challenge 21:

Create the part of a DTD that describes the elements that are pointed to by a menu. The menu offers links to ServiceChat, ContactUs, and PrivacyPolicy

Each element must contain its own Heading, and the text of that heading will appear on the menu, so the heading itself is a reasonable target for a link.

Plus, each element contains a Body, followed by a few optional SeeAlso elements at the end of each section. Each SeeAlso element points to one other element elsewhere on the site.

Challenge 22:

Create the part of the DTD for the elements that will be pointed to by a menu that lists Overview, Description, and Details.

Each element must contain its own Heading element, and each heading will be a target for a link from the menu. In addition to the Heading, each element has a Body, with a few optional SeeAlso links at the end of each section. Each SeeAlso element points to one other element, elsewhere in the site.

Challenge 23:

Create a document that follows the partial DTD above, for CONTENT, with one ARTICLE. The theme is beach trips and vacations on the shore. You do not have much room in the Body: 150 words, max, according to the styleguide.

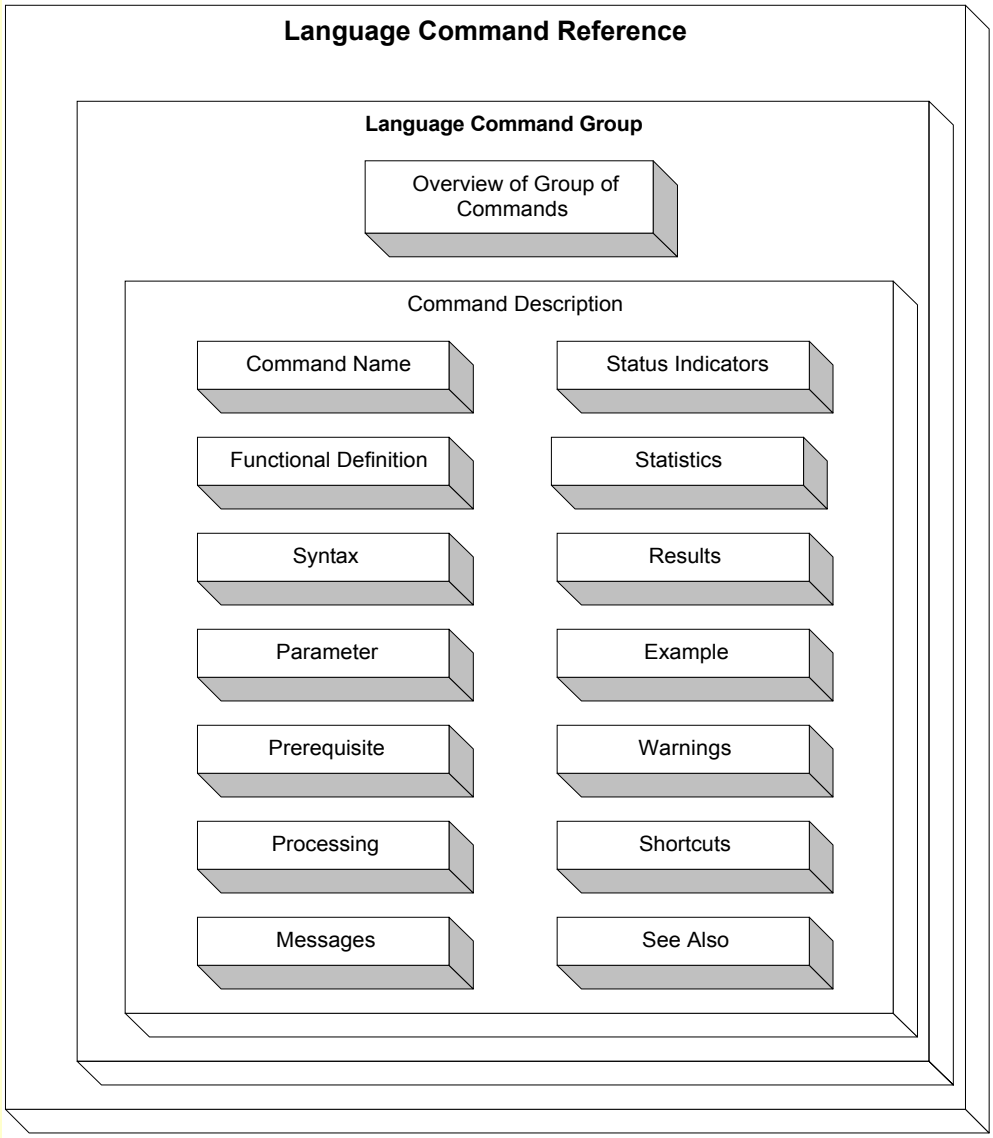
Extended Example

Defining elements and attributes for a command reference:

```
<!ELEMENT reference (languagecommandgroup+)>

<!ELEMENT languagecommandgroup (title, overview,
command.description+)>

<!ELEMENT title (#PCDATA)>

    <!ATTLIST title target ID #REQUIRED>

<!ELEMENT overview (#PCDATA | glossary.term)*>

<!ELEMENT command.description (title, command.name,
command.def, command.syntax, command.param, command.prereq,
command.process, command.messages?, command.status?,
command.stats, command.results, command.example,
command.warnings?, command.shortcuts?, see.also?)>

    <!ATTLIST command.description target ID #REQUIRED>

<!ELEMENT command.name (#PCDATA)>
```

```
<!ELEMENT glossary.term (#PCDATA)>
    <!ATTLIST glossary.term anchor IDREF #REQUIRED>
<!ELEMENT command.def (#PCDATA | glossary.term)*>
    <!ATTLIST command.def target ID #REQUIRED>
<!ELEMENT command.syntax (#PCDATA)>
    <!ATTLIST command.syntax target ID #REQUIRED>
<!ELEMENT command.param (#PCDATA)>
    <!ATTLIST command.param target ID #REQUIRED>
<!ELEMENT command.prereq (#PCDATA | command.prereq.xref |
glossary.term)*>
    <!ATTLIST command.prereq target ID #REQUIRED>
<!ELEMENT command.prereq.xref (#PCDATA)>
    <!ATTLIST command.prereq.xref anchor IDREF #REQUIRED>
<!ELEMENT command.process (#PCDATA | glossary.term)*>
    <!ATTLIST command.process target ID #REQUIRED>
<!ELEMENT command.messages (message.explanation+)>
    <!ATTLIST command.messages target ID #REQUIRED>
<!ELEMENT message.explanation (#PCDATA | message | diagnosis |
glossary.term)*>
    <!ATTLIST message.explanation target ID #REQUIRED>
<!ELEMENT message (#PCDATA)>
<!ELEMENT diagnosis (#PCDATA)>
<!ELEMENT command.status (#PCDATA | glossary.term)*>
    <!ATTLIST command.status target ID #REQUIRED>
<!ELEMENT command.stats (#PCDATA | glossary.term)*>
    <!ATTLIST command.stats target ID #REQUIRED>
<!ELEMENT command.results (#PCDATA | glossary.term)*>
    <!ATTLIST command.results target ID #REQUIRED>
<!ELEMENT command.example (#PCDATA | glossary.term)*>
    <!ATTLIST command.example target ID #REQUIRED>
<!ELEMENT command.warnings (warning.explanation+)>
    <!ATTLIST command.stats target ID #REQUIRED>
<!ELEMENT warning.explanation (#PCDATA | warning |
glossary.term)*>
    <!ATTLIST warning.explanation target ID #REQUIRED>
<!ELEMENT warning (#PCDATA)>
<!ELEMENT command.shortcuts (#PCDATA)>
    <!ATTLIST command.shortcuts target ID #REQUIRED>
<!ELEMENT see.also (#PCDATA | xref)*>
```

```
<!ELEMENT xref (#PCDATA)>
    <!ATTLIST xref anchor IDREF #REQUIRED>
```



This code defines an element called Reference, which includes one or more Language Command Groups. Each Language Command Group has a title, an overview, and one or more command descriptions.

Each command description must include the command name, a functional definition, the command syntax, parameters, prerequisites, a processing description, statistics, results, and an example.

In addition, the command description may have zero or one of each of the following: a description of messages, status indicators, warnings, and possible shortcuts. There may or may not be a single See Also element at the end.

Challenge 24:

Which elements of the command.description are required?

What is included in the message explanation?

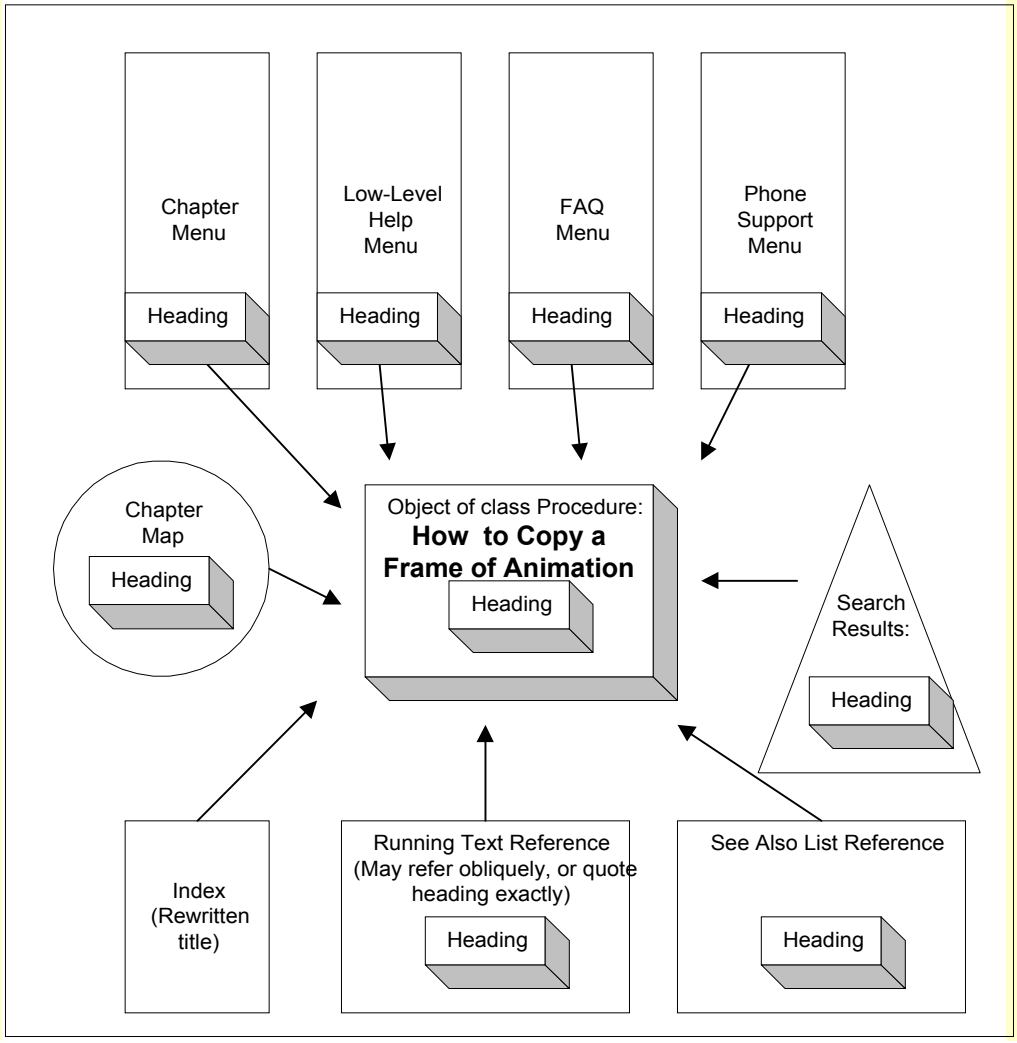
Which elements could serve as pointers to other elements?

6. Reuse elements (and their attributes).

Selected snippets from various element definitions:

```
<!ELEMENT documentation.chapter (chapter.title, chapter.menu,
chapter)>
<!ATTLIST documentation.chapter target ID #REQUIRED>
<!ELEMENT chapter.title (#PCDATA)>
<!ELEMENT chapter.menu (heading+)>
<!ELEMENT heading (#PCDATA)>
<!ATTLIST heading anchor ID #REQUIRED>
<!ATTLIST heading target IDREF #REQUIRED>

<!ELEMENT help.menu (menu.title, heading+)>
<!ATTLIST help.menu target ID #REQUIRED>
<!ELEMENT menu.title (#PCDATA)

<!ELEMENT Faq.menu (menu.title, heading+)>
<!ATTLIST Faq.menu target ID #REQUIRED>

<!ELEMENT phone.menu (menu.title, heading+)>
<!ATTLIST phone.menu target ID #REQUIRED>

<!ELEMENT chapter.map (menu.title, heading+)>
<!ATTLIST chapter.map target ID #REQUIRED>

<!ELEMENT procedure (heading, intro?, step+)>
<!ATTLIST procedure target ID #REQUIRED>

<!ELEMENT index (index.section+)>
<!ATTLIST index target ID #REQUIRED>
<!ELEMENT index.section (index.section.head, index.term+)>
<!ELEMENT index.section.head (#PCDATA)>
<!ATTLIST index.section.head target ID #REQUIRED>
```

```
<!ELEMENT index.term (#PCDATA | heading)*>
<!ATTLIST index.term anchor IDREF #REQUIRED>

<!ELEMENT paragraph (#PCDATA | glossary.term | xref)*>
<!ATTLIST paragraph target ID #REQUIRED>
<!ELEMENT glossary.term (#PCDATA)>
<!ATTLIST glossary.term anchor IDREF #REQUIRED>
<!ELEMENT xref (#PCDATA)>
<!ATTLIST xref anchor IDREF #REQUIRED>

<!ELEMENT see.also (title, heading+)>

<!ELEMENT search.result (description, heading)>
<!ELEMENT description (#PCDATA)>
```



7. Declare entities.

Declaring an entity is like defining a piece of shorthand, a macro, or an auto-text item.

An entity is a box with a label on it: the thing could be a long piece of boilerplate prose, or it could be an object outside of the document, usually some kind of file, information coming from a database, or data generated by another program. An entity is just a bunch of stuff.

When the parser meets an entity in your document, it looks for help to your declaration in the DTD, then it expands the entity, by going to the substitute text or the file, bringing that content back, and dropping it into the page where the entity appeared. We say that it has **included** the entity's content, or that the entity reference has been **replaced** by the entity's content.

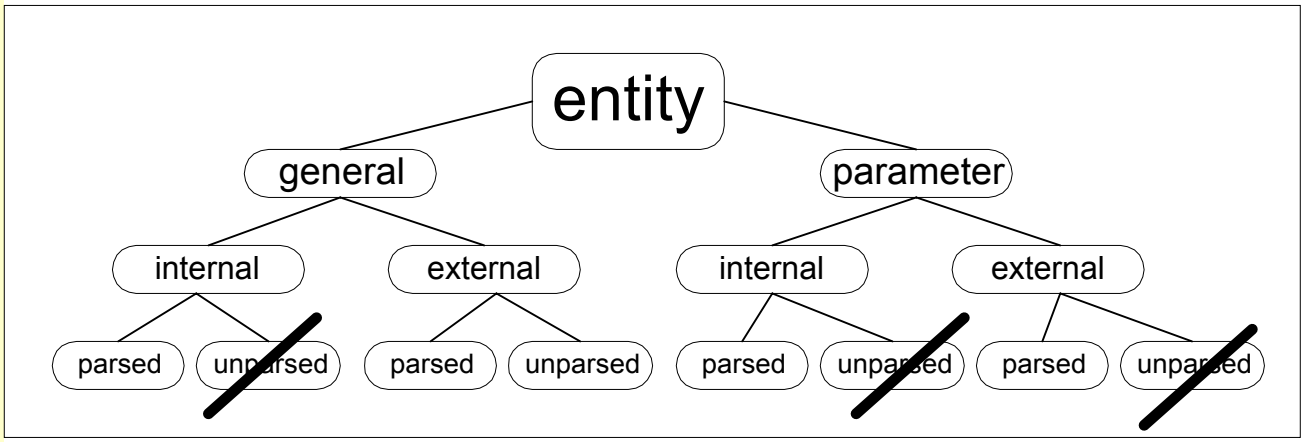
Use entities if…

- You want to put often-used text into a single location, so it will always appear with the same spelling and punctuation throughout your documents.
- You want to reuse modules of your information in many different documents, and intend to identify each one as an entity, to be called when needed.
- You want to keep changeable items such as product names, bundle names, slogans, logos in a single location so that when you update them, the changes spread throughout your document or site, and you do not have to hunt down every instance, to correct each one.
- You have boilerplate such as legal notices, safety warnings, FDA alerts that must be included word for word, and you don't want anyone but lawyers tinkering with them.
- You want to be able to bring in pictures and multimedia materials.

Entities can be external or internal, parsed or unparsed, general or parameter.

- **External entities** exist outside the document. You are pointing to a file or information outside of the document, and asking the parser to fetch it.
- **Internal entities** are declared within the DTD, and appear in a quoted string.
- **Parsed entities** are parsable, that is, they include well-formed content, which is the replacement text, and the parser can review it without hiccuping. You declare the entity in the DTD, giving it a name, and then in the document, use that name as an entity reference. Because they can be parsed, a parsed entity can include markup, such as an element as well as character data.

- **Unparsed entities** are really impossible to parse; they are external files, and they are not XML files. In an element's attribute we use an ENTITY type, and point to the entity. The browser fetches the image or data, and inserts it in that element. The unparsed entity can contain any kind of data, but it is usually not XML data. The parser does not expect to be able to handle this data, but passes along the entity's name to the application such as the browser, which can open the file.
- **General entities** are usable within any XML document. They can be parsed or unparsed; they can include any well-formed content.
- **Parameter entities** can only be used in a DTD.
- **The document itself** is considered the highest level entity. It is the **document entity.** You do not have to declare it, though; in fact, it does not have a name, and cannot be referenced. It is the entity in which the XML declaration and the DOCTYPE declaration take place. In essence, the document is a file.

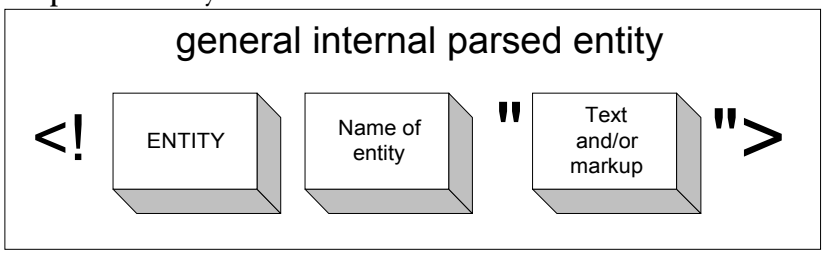


XML does not allow the three types crossed out. As a result, there are only 5 legit entity types.

7a. Declaring a general internal parsed entity

Generally, this is shorthand: an abbreviation or short term for which you provide the full text in the DTD.

- If you want to be able to use the entity in any XML document, you need a **general** entity.
- If you want to declare the entity inside the DTD, giving its replacement text there, you need an **internal** entity.
- If you need to include an element, or some other piece of XML markup, and you want to validate that the replacement text is well-formed, you need a **parsable** entity.



1. Start with <!ENTITY.
2. Give the entity a name, beginning with a letter or an underscore, followed by zero or more letters, digits, periods, hyphens, or underscores.

Oddly, the entity can have the same name as an element or attribute, because that would only confuse a human being.

3. Give the entity a value, in quotes—it's a quoted string, or a literal. The rules:

- You have to surround the value with two apostrophes or two double quotes.
- The value cannot contain the apostrophe or double quotes you are using to delimit the value.
- Your value cannot include the ampersand character except at the beginning of a general entity reference or a character reference. Also, you can't use the percent.
- The value must meet the standards for the spot where you intend to put the entity. That is, if you are going to put it inside an element, it must contain items that can be dropped into that particular element. Or if you intend to put the entity within an attribute, it must meet the standards for attribute values. You may include markup and refer to other entities.

4. Stop with an angle bracket >

Example of a general internal parsed entity:

My initials will stand for my full name.

```
<!ENTITY JP
    "Jonathan Reeve Price">
```

Document:

```
<BIO>
&JP; hails from an adobe house near the Rio Grande.
</BIO>
```

Display:

Jonathan Reeve Price hails from an adobe house near the Rio Grande.

Example of a general internal parsed entity

An element TVSALEITEM can have parsed character data, including entity references. The general internal parsed entity, gizmo, provides some text, and includes the element BLURBLINE.

```
<!ENTITY gizmo
    "The Amazing Gizmo
<!BLURBLINE>It slices. It dices. It cleans windows.</
BLURBLINE>">
```

Document

```
<TVSALEITEM>
&gizmo;
Buy this today for 10 percent off.
</TVSALEITEM>
```

Display:

```
The Amazing Gizmo.
It slices.  It dices.  It cleans windows.
Buy this today for 10 percent off.
```

Challenge 25:

Which of the following could be declared as a general internal parsed entity?

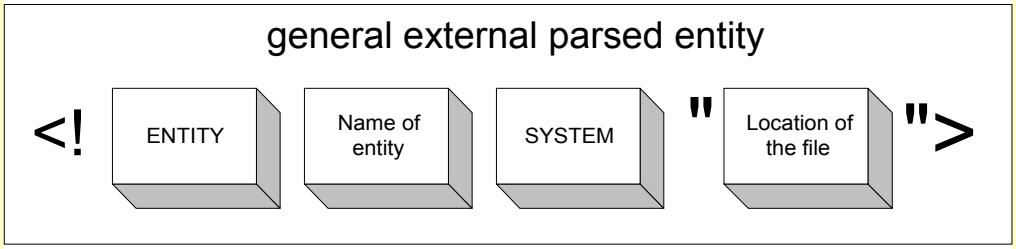
- A piece of boilerplate that you want to ensure will remain unchanged wherever it appears.

- A standard text that includes an element.
- An image stored on the same server as the DTD.
- A stretch of text that you do not want parsed, because it must include characters that would disturb the parser.

7b. Declaring a general external parsed entity

Generally, this is a pointer to an external file that can be successfully parsed.

- If you want to be able to use the entity in any XML document, you need a **general** entity.
- If you want to point to an external file that contains items that can legally be inserted into an element (characters, nested elements, so on), you need an **external** entity.
- If you need to include an element, or some other piece of XML markup, and you want to validate that the replacement text is well-formed, you need a **parsable** entity.



The location gives a full URL, or a relative URL (if in the same directory).

Example of a general external parsed entity

We point to another document called Benefits.xml.

```
<!ELEMENT VIDEO ANY>

<!ENTITY BEN SYSTEM "http://www.theprices.com/benefits.xml">
```

Document

```
<VIDEO>

This great new exercise video has tons of benefits for health-conscious teens:
```

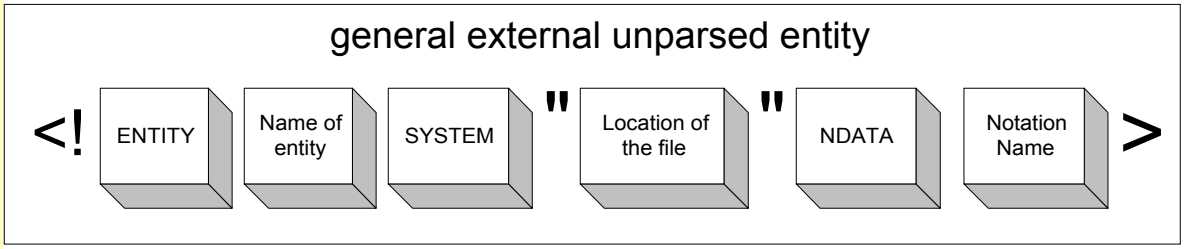
```
&BEN;
</VIDEO>
```

Caution: the XML spec does not require the parser to validate an XML document. If the processor does not validate, it ignores declarations of external parsed entities, whether general or parameter. They show up only as links, for the user.

7c. Declaring a general external unparsed entity

This is a pointer to an external file that can **NOT** be successfully parsed. That file can contain any kind of data, as long as it matches the description you give when you set up a NOTATION for it.

- If you want to be able to use the entity in any XML document, you need a **general** entity.
- If you want to point to an external file that contains items that can legally be inserted into an element (characters, nested elements, so on), you need an **external** entity.
- If the other file contains material that cannot be parsed, such as an image, video, or sound, you are dealing with an **unparsable** entity. Material that cannot be parsed cannot go in the DTD, so it is general; and must be outside the DTD, so it is external. So unparsed implies the other two terms.



1. Start with <!ENTITY

2. State the name of the entity.

3. Put in SYSTEM to indicate you are about to give a specific location for the external file.

4. In single or double quotes put the location. (The system literal).

   The location gives a full URL, or a relative URL (if in the same directory).

5. Add NDATA to warn the parser that the file contains

unparsable data.

6. Put in the name of the notation that file uses (its file format or application, previously defined as a NOTATION).

Example of a general external unparsed entity

We want to refer to a gif file with the logo for a Non-Governmental Organization.

```
<!ELEMENT NGO (NAME, DESCRIPTION, LOGO)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT LOGO EMPTY)>
<!ATTLIST LOGO Source ENTITY #REQUIRED>

<!NOTATION GIF SYSTEM "ShowGif.exe">
<!ENTITY Ruckuslogo SYSTEM "http://www.theprices.com/
Ruckus.gif" NDATA GIF>
```

Document

```
<NGO>
<NAME>Santa Fe Growers Coop</NAME>
<DESCRIPTION>A cooperative bringing together consumers and
farmers in the Santa Fe area, producing farmers" markets and
wholesale selling to local groceries.</DESCRIPTION>
<LOGO Source="&Ruckuslogo;"/>
</NGO
```

Challenge 26:

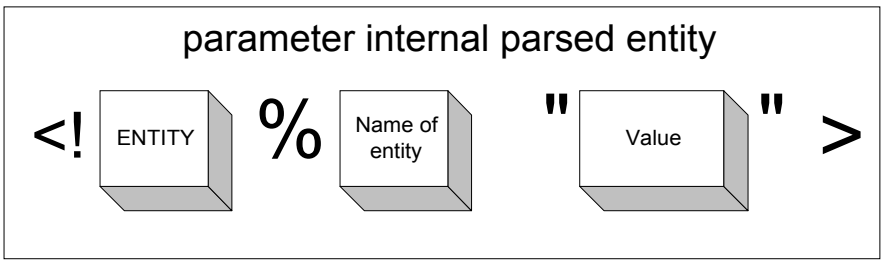
Which of the following should be declared as a general external parsed entity, and which should be considered a general external unparsed entity?

- ☐ A video clip.
- ☐ A file containing raw text plus an element marked up in XML.
- ☐ An image in png format.
- ☐ A text file containing company names.
- ☐ Another page of XML.

7d. Declaring a parameter internal parsed entity

Sometimes you want to use the same markup text in several different places within a DTD. This special kind of entity is thought of as a parameter of the DTD, hence the ugly name *parameter entity*. It includes markup declarations.

- If you want to be able to use the entity in a DTD, you need a **parameter** entity.
- If you are going to declare the entity within the DTD, you need an **internal** entity.
- If the entity contains material that can be parsed, you are dealing with an **parsable** entity. Because parameter entities aim to go in DTDs, they must be parsable.



1. Start with <!ENTITY.

2. Insert a percent sign %.

3. Give the entity a name.

The name must begin with a letter or an underscore, followed by zero or more letters, digits, periods, hyphens, or underscores.

The entity can have the same name as an element or attribute or a general entity, because that would only confuse a human being. On the other hand, for humans, try something different.

4. Give the entity a value, in quotes—it's a quoted string, or a literal. The rules:

- You have to surround the value with **two apostrophes** or **two double quotes**.
- The value **cannot** contain the apostrophe or double quotes you are using to delimit the value.
- Your value cannot include the **percent** sign. You cannot use the **ampersand** character except at the beginning of a general entity reference or a character reference.
- The value must meet the standards for the spot where you intend to put the entity. You can put this entity into the DTD, but not inside some other markup declaration. So it can contain element type declarations, attribute list declarations, general entity declarations, notation declarations, processing

instructions, or comments. You cannot include a parameter declaration that appears later in the DTD and its reference. (There are a few exceptions to this general rule. See Section 4 of the XML spec at http://www.w3.org/TR/REC-xml).

5. Stop with an angle bracket >

Example of a parameter internal parsed entity:

```
<!ENTITY %team
"<!- - members of the team - ->
<!ELEMENT MEMBER (#PCDATA)>
    <!ATTLIST MEMBER department CDATA 'marcomm'>"
>
…
<!ELEMENT ZEBRA (#PCDATA)>
%team;
```

The last two lines are the equivalent of:

```
<!ELEMENT ZEBRA (#PCDATA)>
<!- - members of the team - ->
<!ELEMENT MEMBER (#PCDATA)>
    <!ATTLIST MEMBER department CDATA "marcomm">
```

Challenge 27:

Create a parameter internal parsed entity to be used in alerting developers to the status of an element being declared within the DTD. The entity is called TK and contains a comment saying, "Incomplete declaration. The specs are still to come."

Challenge 28:

Create a parameter internal parsed entity to be used in identifying the status of an element being declared within the DTD. The entity is called TBD and contains a comment saying, "To be determined. This dummy element is just a placeholder, for now."

When using a parameter internal parsed entity in your DTD, you precede it with the **percentage sign**, and end with the **semicolon**.

Example from the DTD for XHTML:

```
<!- - Core attributes common to most elements

id     document-wide unique id

class     space separated list of classes

style     associated style info

title     advisory title/ amplification - ->

<!ENTITY %coreattrs

"id ID #IMPLIED

class CDATA #IMPLIED

style %StyleSheet; #IMPLIED

title %Text; #IMPLIED" >
```

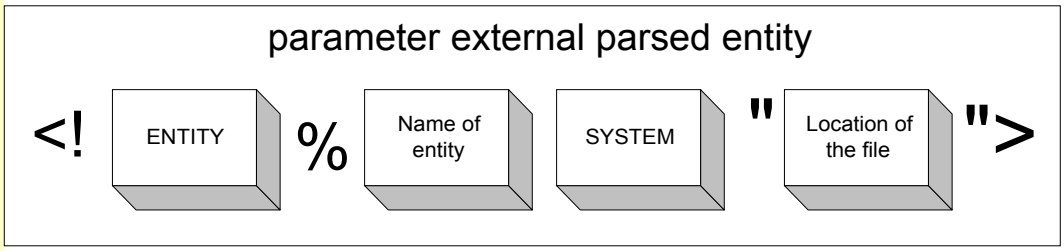
From now on, whenever the team wants to drop in the core attributes, instead of typing them all out, using the entity %coreattrs; will tell the software to put them in automatically.

Note: Two entities, %StyleSheet; and %Text; have already been defined, and are here reused. To make cross reference easier, the W3C turns each of these entities into a link, so you can click to find out what it stands for.

7e. Declaring a parameter external parsed entity

You are pointing to a file that contains complete markup declarations of the types allowed in a DTD, which will be drawn back into the current DTD. You might use this entity if you create a series of DTDs for different departments, and sometimes put together two or three, and sometimes a different set. In that case, you would build several DTDs using parameter external parsed entities to call whatever files you need, essentially assembling several DTDs as needed.

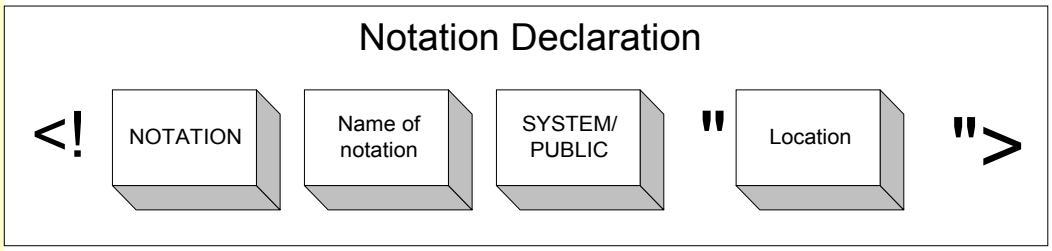
- If you want to be able to use the entity in the DTD, you need a **parameter** entity.
- If you want to point to an external file that contains items that can legally be inserted into a DTD, you need an **external** entity.
- To be usable within the DTD, the content of this external file must be **parsable**.



Note: parameter entities cannot be used in the document itself. They just show up as weird text beginning with a percent sign.

General entity references almost always appear in the document. One exception: if an attribute declaration includes a general entity in the DTD.

8. Declare a notation to identify the file formats of external unparsed entities.



Name the file format so you can refer to it in attributes, unparsed entity declarations, and processing instructions.

These are times when you may need to alert an application or the parser what kind of file you are pointing to, or hoping to have processed.

Because we do not want the XML parser to bother with these files, we call them **unparsed entities**. Really, we should call them **unparsable**, but that would be unpronounceable.

The rest of the declaration points to the documentation on that file format, a formal spec, or, best, an application that can handle objects that have been created in this notation.

1. Start with <!NOTATION

2. Give your notation a name.

3. Say whether the location is SYSTEM or PUBLIC.

If you are giving a specific location, use SYSTEM. If you believe the application can find the well-known location, use PUBLIC, and cross your fingers.

4. For SYSTEM, give one location. For PUBLIC, give the public location, then, just in case, give a specific location, too. Put the location(s) in quotes.

5. Close with angle bracket >

Example of Notation declaration

If you expect to point to a Word document, you might mention Ms Word, as a "helper application."

```
<!NOTATION doc SYSTEM "http://www.theprices.com/word.exe">
```

Example of notation for date formats

If you have a date element that may have either ISO or EU date formats…

```
<!NOTATION ISODATE SYSTEM "http://www.iso.ch/
date_specification">
<!NOTATION EUDATE SYSTEM "http://www.eu.eu/
date_specification">
<!ELEMENT TODAYSDATE (#PCDATA)>
    <!ATTLIST TODAYSDATE date-format NOTATION (ISODATE |
    EUDATE) #REQUIRED)>
```

Challenge 29:

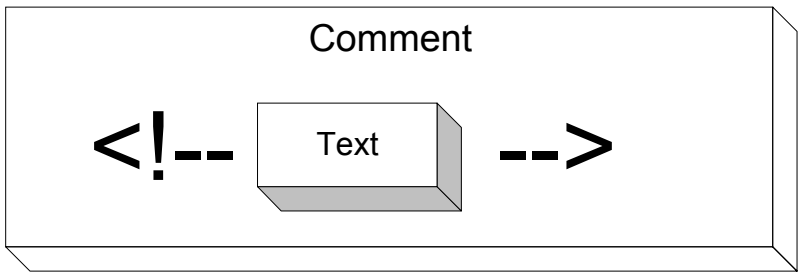
Create a PHOTO element that will be receiving images in either jpg or bmp formats.

Challenge 30:

Create a STANDARD element that will be receiving files in doc, htm, html, txt formats.

9. Add comments for humans, only.

Use comments for headings or notes to yourself, your team, or some other human being. Once you are inside a comment, you can write any darn thing you want. The XML parser ignores the comments. The comment is set up just as it would be within an XML document.



9a. Start the comment with <!--.

9b. Write anything you want.

9c. End the comment with -->.

Caution: Do not try to put a double hyphen inside the comment!

```
<!- -File Name: booklist.xml- ->
<!--This is a first draft for internal circulation only.-->
<!--Date: July 4, 2001 -->
```

Super Challenge: Writing to a DTD

Following this DTD, produce a draft of the catalog, containing only one product description—for a table saw named The Ferret Table Saw, part number T765, from the Milwaukee plant, currently back-ordered.

The saw weighs 532 pounds; uses 220 volt current; and meets OSHA standards.

The options include the finish, which is Metal, but no adapter. There is no case.

The Manufacturer's Suggested Retail Price (MSRP) is $1577. The wholesale price is $835. Average street price is $1199. Shipping costs average $185. Note that this is an excellent buy.

Please make up locations for the DTD and the stylesheet.

```
<!DOCTYPE CATALOG [
<!ELEMENT CATALOG (PRODUCT+)>
<!ELEMENT PRODUCT (DESCRIPTION, SPECIFICATIONS+, OPTIONS?, PRICE+, NOTES?)>
<!ATTLIST PRODUCT NAME CDATA #IMPLIED>
<!ATTLIST PRODUCT CATEGORY (HandTool | Table | Shop-Professional) "HandTool">
<!ATTLIST PRODUCT PARTNUM CDATA #IMPLIED>
<!ATTLIST PRODUCT PLANT (Pittsburgh | Milwaukee | Chicago) "Chicago">
<!ATTLIST PRODUCT INVENTORY (InStock | Backordered | Discontinued) "InStock">
<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS WEIGHT CDATA #IMPLIED>
<!ATTLIST SPECIFICATIONS POWER CDATA #IMPLIED>

<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS FINISH (Metal | Polished | Matte) "Matte">
<!ATTLIST OPTIONS ADAPTER (Included | Optional | NotApplicable) "Included">
<!ATTLIST OPTIONS CASE (HardShell | Soft | NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE MSRP CDATA #IMPLIED>
<!ATTLIST PRICE WHOLESALE CDATA #IMPLIED>
<!ATTLIST PRICE STREET CDATA #IMPLIED>
<!ATTLIST PRICE SHIPPING CDATA #IMPLIED>
<!ELEMENT NOTES (#PCDATA)>
```

<!ENTITY AUTHOR “Bob Ferret”>

<!ENTITY COMPANY “Ferret Equipment”>

<!ENTITY EMAIL “rf@ferretequipment.com”>

Adapted from http://www.vervet.com/

Grand Challenge: Analyzing a document to create a DTD

A DTD is an ideal picture of one type of document. It tells you what components must be there, which are optional, and how often.

Here is a sample document, followed by a diagram showing all the components the team figures it would ever put into a procedure. Please create a DTD for this type of document. Remember: the example is just one variation on the type. The diagram shows the abstract view, a standard procedure that includes all possible components, and shows their relationships.

Sample Document:

Getting a close-up view of your document

If you want to enlarge the text onscreen, so you can check those commas, you can pick a particular percentage of the regular view, enlarging that to 150% or more.

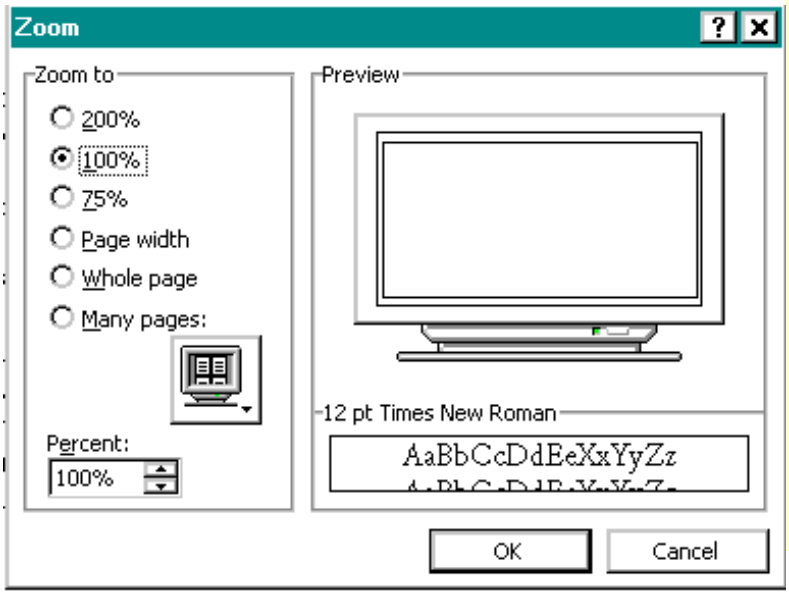
You must be in Page Layout or Outline view. (Choose these from the View menu).

You need a mouse for this procedure.

The key concept here is zooming in, to see everything bigger. You can also zoom out, to reduce the size of the text, but scan more of it at once. (See our next procedure, "Getting a bird's eye view of your document").

1.Choose View => Zoom.

You see the Zoom dialog, which lets you pick the way you want to enlarge (or shrink) the document onscreen.



2. On the left side of the dialog, click 200%, or if you want a different enlargement, click the up arrow in the Percent panel.

Warning: do not choose Whole Page or Many Pages, because your document will suddenly seem to shrink onscreen, to fit the whole page, or several pages onto the display.

Percentage means the amount that you want to change the size of the document onscreen. Percentages above 100 make it look bigger; percentages below 100% make it smaller.

3. Click OK.

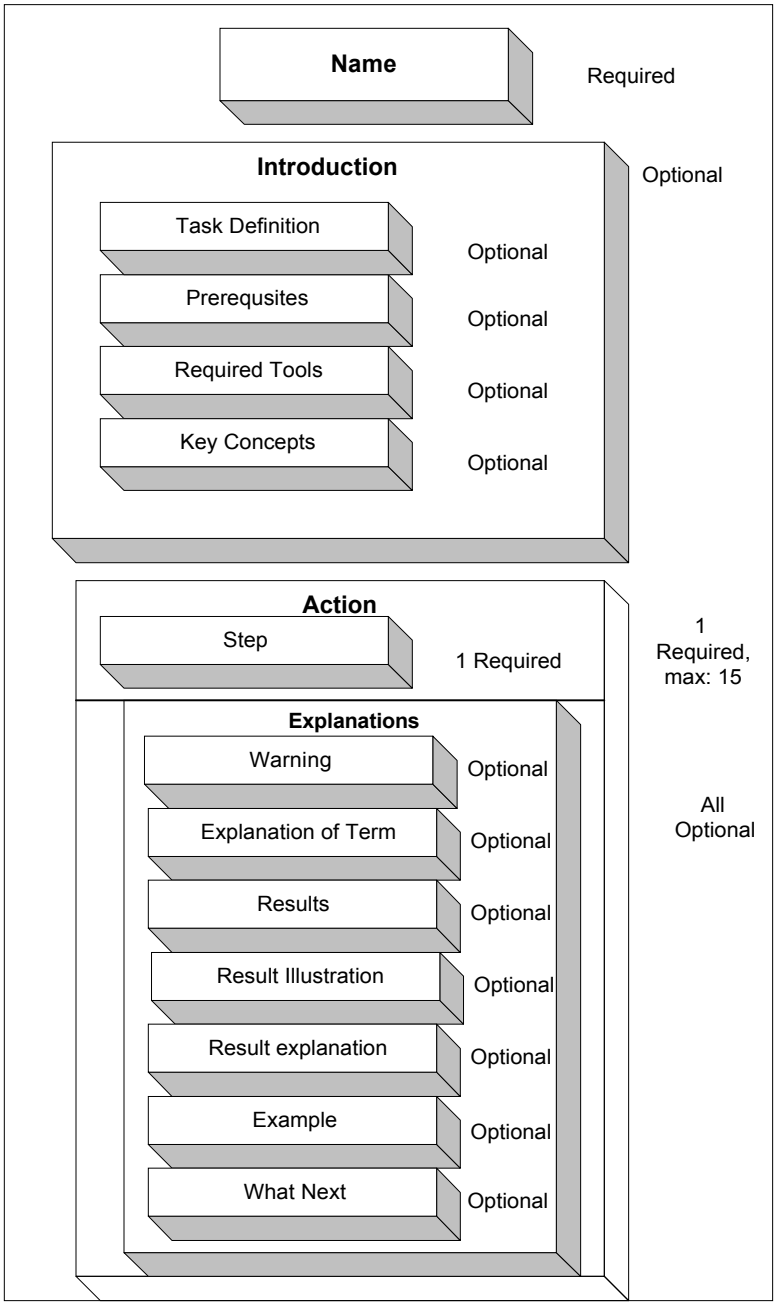
The result: your document looks a lot bigger.

You have zoomed out.

Example: Imagine you have some text in italic, and wonder whether or not that thing at the end of the sentence is a comma or a period. At normal size, with small type, you can"t tell. So you decide to enlarge it. You choose Zoom on the View menu, and pick 200%. Ah, now you can tell: it is a comma. You can make the change while zoomed in, then go back to the normal view, confident that you are ending your sentences with periods.

Next: shrinking the document onscreen.

Diagram of procedure



Use your DTD and model to identify and reorganize existing material.

Here is a step with scrambled explanatory material. Using the DTD you have just created, and the diagram of an abstract procedure, identify the explanatory elements, and draw arrows showing how this should be reorganized to follow the ideal pattern. If necessary, rewrite as you go.

2. Scroll to the end of your document.

The twenty-page document extends out below this window. The vertical bar on the right of your window is known as the scroll bar because it offers several ways to scroll through your document.

Imagine that your document is a giant papyrus scroll, and your slave is unrolling it behind the window, so you can only see a little bit of a time.

To move up and down in the document quickly, you have to drag the little rectangle in the scroll bar, or press the up or down arrow.

Do not press the double-down-arrow at this time because that will just take you to the next heading. The scroll bar looks like this: (art). The position of the elevator, that rectangle in the scroll bar, shows you about how far through the document you are now.

For example, if you are at the beginning of the document, your elevator is at the top; if you wanted to go to the middle, you would drag it to the middle of the scroll bar. If you have carried out our instruction, you are now at the end of the document, because your elevator has reached the bottom of the scroll bar, just above the down-pointing arrow at the bottom.

Reading a DTD for fun

```
<!- - DTD for Shakespeare    J. Bosak    1994.03.01, 1997.01.02 - - >

<!- - Revised for case sensitivity 1997.09.10 - - >

<!- - Revised for XML 1.0 conformity 1998.01.27 (thanks to Eve Maler) -
- >

<!ENTITY amp "&#38;">

<!ELEMENT PLAY     (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT,
INDUCT?, PROLOGUE?, ACT+, EPILOGUE?)>

<!ELEMENT TITLE    (#PCDATA)>

<!ELEMENT FM       (P+)>

<!ELEMENT P        (#PCDATA)>

<!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>

<!ELEMENT PGROUP   (PERSONA+, GRPDESCR)>

<!ELEMENT PERSONA  (#PCDATA)>

<!ELEMENT GRPDESCR (#PCDATA)>

<!ELEMENT SCNDESCR (#PCDATA)>

<!ELEMENT PLAYSUBT (#PCDATA)>

<!ELEMENT INDUCT   (TITLE, SUBTITLE*,
(SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>

<!ELEMENT ACT      (TITLE, SUBTITLE*, PROLOGUE?, SCENE+,
EPILOGUE?)>

<!ELEMENT SCENE    (TITLE, SUBTITLE*, (SPEECH | STAGEDIR |
SUBHEAD)+)>

<!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>

<!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>

<!ELEMENT SPEECH   (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>

<!ELEMENT SPEAKER  (#PCDATA)>

<!ELEMENT LINE     (#PCDATA | STAGEDIR)*>

<!ELEMENT STAGEDIR (#PCDATA)>

<!ELEMENT SUBTITLE (#PCDATA)>

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

Outtakes from dream.xml

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

<!DOCTYPE PLAY SYSTEM "play.dtd">

<PLAY>

<TITLE>A Midsummer Night's Dream</TITLE>

<FM>

<P>Text placed in the public domain by Moby Lexical Tools, 1992.</P>
```

```
<P>SGML markup by Jon Bosak, 1992-1994.</P>
<P>XML version by Jon Bosak, 1996-1998.</P>
<P>This work may be freely copied and distributed worldwide.</P>
</FM>
```

Last scene of the play

```
<STAGEDIR>Enter OBERON and TITANIA with their train</
STAGEDIR>
<SPEECH>
<SPEAKER>OBERON</SPEAKER>
<LINE>Through the house give gathering light,</LINE>
<LINE>By the dead and drowsy fire:</LINE>
<LINE>Every elf and fairy sprite</LINE>
<LINE>Hop as light as bird from brier;</LINE>
<LINE>And this ditty, after me,</LINE>
<LINE>Sing, and dance it trippingly.</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>TITANIA</SPEAKER>
<LINE>First, rehearse your song by rote</LINE>
<LINE>To each word a warbling note:</LINE>
<LINE>Hand in hand, with fairy grace,</LINE>
<LINE>Will we sing, and bless this place.</LINE>
</SPEECH>
<STAGEDIR>Song and dance</STAGEDIR>
<SPEECH>
<SPEAKER>OBERON</SPEAKER>
<LINE>Now, until the break of day,</LINE>
<LINE>Through this house each fairy stray.</LINE>
<LINE>To the best bride-bed will we,</LINE>
<LINE>Which by us shall blessed be;</LINE>
<LINE>And the issue there create</LINE>
<LINE>Ever shall be fortunate.</LINE>
<LINE>So shall all the couples three</LINE>
<LINE>Ever true in loving be;</LINE>
<LINE>And the blots of Nature's hand</LINE>
<LINE>Shall not in their issue stand;</LINE>
```

<LINE>Never mole, hare lip, nor scar,</LINE>
<LINE>Nor mark prodigious, such as are</LINE>
<LINE>Despised in nativity,</LINE>
<LINE>Shall upon their children be.</LINE>
<LINE>With this field-dew consecrate,</LINE>
<LINE>Every fairy take his gait;</LINE>
<LINE>And each several chamber bless,</LINE>
<LINE>Through this palace, with sweet peace;</LINE>
<LINE>And the owner of it blest</LINE>
<LINE>Ever shall in safety rest.</LINE>
<LINE>Trip away; make no stay;</LINE>
<LINE>Meet me all by break of day.</LINE>
</SPEECH>
<STAGEDIR>Exeunt OBERON, TITANIA, and train</STAGEDIR>
<SPEECH>
<SPEAKER>PUCK</SPEAKER>
<LINE>If we shadows have offended,</LINE>
<LINE>Think but this, and all is mended,</LINE>
<LINE>That you have but slumber'd here</LINE>
<LINE>While these visions did appear.</LINE>
<LINE>And this weak and idle theme,</LINE>
<LINE>No more yielding but a dream,</LINE>
<LINE>Gentles, do not reprehend:</LINE>
<LINE>if you pardon, we will mend:</LINE>
<LINE>And, as I am an honest Puck,</LINE>
<LINE>If we have unearned luck</LINE>
<LINE>Now to 'scape the serpent's tongue,</LINE>
<LINE>We will make amends ere long;</LINE>
<LINE>Else the Puck a liar call;</LINE>
<LINE>So, good night unto you all.</LINE>
<LINE>Give me your hands, if we be friends,</LINE>
<LINE>And Robin shall restore amends.</LINE>
</SPEECH>
</SCENE>
</ACT>
</PLAY>

Answers

Challenge 0

```
<!ELEMENT Memo (To, CC?, BCC?, From, Date, Re, Body)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT BCC (#PCDATA)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Re (#PCDATA)>
<!ELEMENT Body (#PCDATA)>
```

Challenge 1

```
<!ELEMENT Report (author, title,
category, abstract, keywords,
heading1?, heading 2, paragraph)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ELEMENT heading1 (#PCDATA)>
<!ELEMENT heading2 (#PCDATA)>
<!ELEMENT paragraph(#PCDATA)>
```

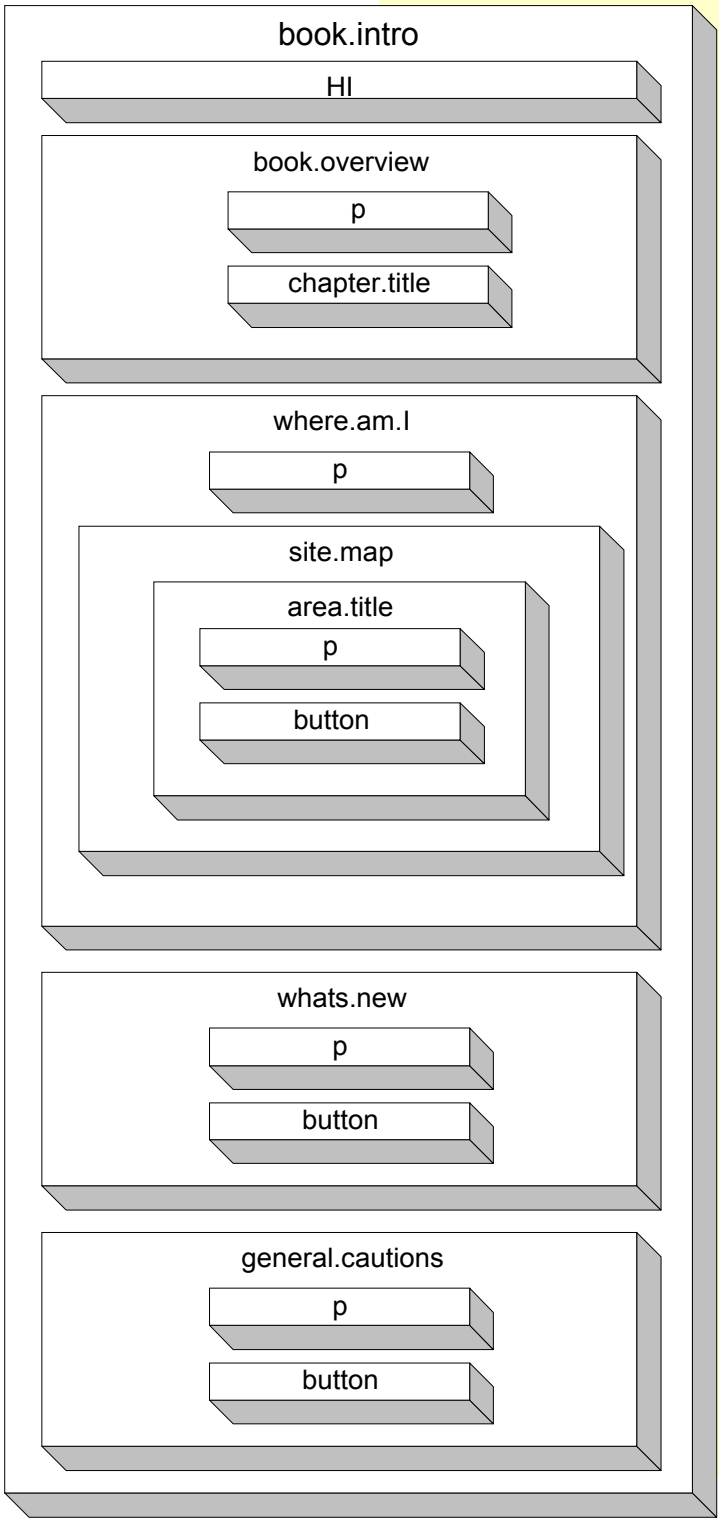
Challenge 2

This element is not valid, because the label element is in the wrong place, and the cd_description element is missing.

Challenge 3

See diagram =>.

Challenge 3: diagram



Challenge 4

For one of three sizes (no other sizes allowed): Enumerated values.

A Social Security Number: NMTOKEN.

The name of an image file, previously defined as an entity within your DTD: ENTITY.

An indication of file type, chosen from several already defined as notations: NOTATION.

A unique all-text password: ID.

One or more elements, which you want to refer to: IDREFS

Challenge 5:

Which type would you choose for the following attributes?

One of three kinds of book cover. Enumerated values.

A serial number for each product. ID.

A pointer to a product, using its serial number. IDREF

A code that starts with an underscore. ID

The name of an entity that describes an external file. ENTITY

The date format being used. NOTATION

Challenge 6

<!ELEMENT PRODUCT (#PCDATA)>

<!ATTLIST PRODUCT DESCRIPTION CDATA #IMPLIED>

<!ATTLIST PRODUCT Price CDATA #IMPLIED>

Challenge 7

<!ELEMENT Partner (#PCDATA)>

<!ATTLIST Partner MISSION CDATA #IMPLIED

HEADQUARTERS_CITY CDATA #IMPLIED>

Challenge 8

<!ELEMENT employee (#PCDATA)>

<!ATTLIST employee STATUS (fulltime | parttime | contractor) #IMPLIED>

Challenge 9

<!ELEMENT AGENT (#PCDATA)>

<!ATTLIST AGENT SECURITY (None | Secret | Top_Secret | Eyes_Only) #IMPLIED>

Challenge 10

<!ELEMENT COURSE (#PCDATA)>

<!ATTLIST COURSE code ID #REQUIRED>

Challenge 11

<!ELEMENT CONTRACT (#PCDATA)>

<!ATTLIST CONTRACT contract_number ID #REQUIRED>

Challenge 12

```
<!ELEMENT HOWTO (#PCDATA)>
<!ATTLIST HOWTO relevant.tips IDREFS #IMPLIED>
<!ELEMENT TIP (#PCDATA)>
<!ATTLIST TIP TIP_CODE ID #REQUIRED>
```

Challenge 13

```
<!ELEMENT SCENT (#PCDATA)>
<!ATTLIST SCENT products_with_this_fragrance IDREFS #IMPLIED>
<!ELEMENT PRODUCT (NAME, PACKAGE, SIZE, LANGUAGE,
SCENT)>
<!ATTLIST PRODUCT sku ID #REQUIRED>
…
```

Challenge 14

```
<!ELEMENT PART (#PCDATA)>
<!ATTLIST PART creation_date NMTOKEN #REQUIRED>
```

Challenge 15

```
<!ELEMENT CONFIRMATION (#PCDATA)>
<!ATTLIST CONFIRMATION message_number ID #REQUIRED>
```

Challenge 16

```
<!ELEMENT INVENTORY (ITEM*)>
<!ELEMENT ITEM (#PCDATA)>
<!ATTLIST ITEM
    Code ID #REQUIRED
    Description CDATA #IMPLIED
    Price NMTOKEN #REQUIRED
    Suggestion IDREFS #IMPLIED
    Featurelist IDREF #REQUIRED
```

Code: a unique identifier

Description: rambling raw text

Price: begins with a number

Suggestion: there may be a number of tips referenced here.

Featurelist: this is another element we are pointing them to, through a link.

Problems:

In both items, the code has a value beginning with a number, even though that is invalid for an ID.

The first item has no reference to a Featurelist, which is required.

The second item does not follow the order in which the attributes were defined, but that is OK for the software. It just drives humans nuts.

Challenge 17

In DTD:

<!ELEMENT LOGO EMPTY>

<!ATTLIST LOGO source ENTITIES #REQUIRED>

In document:

<LOGO source=" OurLogosmall" />

Challenge 18

In DTD:

<!ELEMENT WHITEPAPER EMPTY

<!ATTLIST WHITEPAPER pointer ENTITY #REQUIRED>

In document:

<WHITEPAPER pointer="WhitePaper001" />

Challenge 19

In DTD:

<!ELEMENT Face EMPTY>

<!ATTLIST Face

SSN NMTOKEN #REQUIRED

student_number NMTOKEN #REQUIRED

source ENTITY #REQUIRED

fileformat (png | gif | jpg) #REQUIRED>

In document:

<Face SSN="555-55-5555" student_number="54321" source=JPRICE fileformat="jpg" />

Challenge 20

<!ELEMENT Painting EMPTY>

<!ATTLIST Painting

accession_number NMTOKEN #REQUIRED

inventory_number NMTOKEN #REQUIRED

source ENTITY #REQUIRED

file_format (bmp | tif | png | gif | jpg) #REQUIRED >

In document:

<Painting accession_number="20030328" inventory_number="10789" source ="GUERNICA" file_format = "bmp" />

Challenge 21

<!ELEMENT ServiceChat (Heading, Body, SeeAlso*)>

<!ELEMENT ContactUs (Heading, Body, SeeAlso*)>

<!ELEMENT PrivacyPolicy (Heading, Body, SeeAlso*)>

<!ELEMENT Heading (#PCDATA)>

<!ATTLIST Heading target ID #REQUIRED>

<!ELEMENT Body (#PCDATA)>

<!ELEMENT SeeAlso (#PCDATA)>

<!ATTLIST SeeAlso reference IDREF #REQUIRED>

Challenge 22

<!ELEMENT Overview (Heading, Body, SeeAlso*)>

<!ELEMENT Description (Heading, Body, SeeAlso*)>

<!ELEMENT Details (Heading, Body, SeeAlso*)>

<!ELEMENT Heading (#PCDATA)>

<!ATTLIST Heading target ID #REQUIRED>

<!ELEMENT Body (#PCDATA)>

<!ELEMENT SeeAlso (#PCDATA)>

<!ATTLIST SeeAlso reference IDREF #REQUIRED>

Challenge 23

<CONTENT>

<ARTICLE AUTHOR="JONATHAN PRICE" EDITOR="LISA PRICE" DATE ="20040606" EDITION="MORNING" ARTICLE_CODE = "M335">

<HEADLINE ID="H335">Big Beach on the Big Island </HEADLINE>

<BYLINE>By Jonathan Price

<EMAIL biolink="Contact33"></BYLINE>

<LEAD>Black lava sands, where Captain Cook was killed, lead you past ancient altars. </LEAD>

<BODY> The beaches on Hawaii, the biggest of the islands, let you tip your toe into history. One cove surrounds an altar to Lono, the shark god, where enemy warriors were tied on a pile of rocks, waiting for the tide to come in, with the sharks, to devour the freshly drowned sacrificial victims. On the Black Sand Beach, you can walk where Captain Cook quarreled with the locals over petty thievery; they took his life, and drove away his boats.

</BODY>

<NOTES> For more on history underfoot, see http://www.melekalikimaka.org.</NOTES>

</ARTICLE>

</CONTENT>

Challenge 25

Boilerplate: Yes, unless it contains illegal characters.

A standard text including an element: Yes.

An image: no, this file is external.

A stretch of text that you do not want parsed: Well, then you do not want it to be parsed.

Challenge 26:

A video clip: General External Unparsed Entity

A file containing raw text plus an element marked up in XML: General External Parsed Entity

An image in png format: General External Unparsed Entity

A text file containing company names: General External Parsed Entity

Another page of XML: General External Parsed Entity

Challenge 27

<!ENTITY %TK

"<!- - Incomplete declaration. The specs are still to come.- ->" >

Challenge 28

<!ENTITY %TBD

"<!- - To be determined. This dummy element is just a placeholder, for now.- ->">

Challenge 29

<!ELEMENT PHOTO EMPTY>

<!ATTLIST PHOTO

source ENTITY #REQUIRED

file_type (jpg | bmp) #REQUIRED>

Challenge 30

<!ELEMENT STANDARD EMPTY>

<!ATTLIST STANDARD

source ENTITY #REQUIRED

file_type (doc | htm | html | txt) #REQUIRED>

Super Challenge

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

<!DOCTYPE CATALOG SYSTEM http://www.theprices.com/dtd/catalog.dtd [<!ENTITY tablesawimage SYSTEM http://www.theprices.com/images/tablesaw.jpg NDATA JPG>]>

<?xml-stylesheet type="text/xsl" href="http://www.theprices.com/catalog.xsl"?>

<CATALOG>

<PRODUCT NAME="The Ferret Table Saw" CATEGORY = "Table" PARTNUM ="t4765" PLANT = "Milwaukee" INVENTORY = "Backordered">

<DESCRIPTION>A magnificent addition to any shop. Cuts wood up to one-foot thick. 15 interchangeable blades.

&tablesawimage;

</DESCRIPTION>

<SPECIFICATIONS WEIGHT="532" POWER="220"> Weight: 532 pounds.

Power: 220 volts.

Meets OSHA standards.

</SPECIFICATIONS>

<OPTIONS FINISH="Metal" ADAPTER="NotApplicable" CASE="NotApplicable"> The finish is an excellent gun-metal.</OPTIONS>

<PRICE MSRP="1577" WHOLESALE="835" STREET= "1199" SHIPPING="185">Manufacturer's Suggested Retail Price: $1577</PRICE>

<NOTES>An excellent buy. &AUTHOR; &COMPANY; &EMAIL;</NOTES>

</PRODUCT>

</CATALOG>