Process, Power and Marine Division

SmartPlant 3D Reports Workshop
Intro to XML

May, 2005



INTERGRAPH

What is XML?

XML is a **markup language** for documents containing **structured information**.

A **markup language** is a mechanism to identify structures in a document.

The XML specification defines a standard way to add markup to documents.

Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

What Do XML Documents Look Like? Simple Example

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

<oldjoke>

<burns>Say <quote>goodnight</quote>,
Gracie.</burns>

 <allen><quote>Goodnight,
Gracie.</quote></allen>

<applause/>

</oldjoke>
```

What Do XML Documents Look Like? Declaration

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

<oldjoke>

<burns>Say <quote>goodnight</quote>,
Gracie.</burns>

 <allen><quote>Goodnight,
Gracie.</quote></allen>

<applause/>

</oldjoke>
```

XML Declaration, Not Required, explicitly states XML Document and Version

XML – Markups and Content

- 6 kinds of markups
  - **Elements**
  - **Attributes**
  - **Entity References**
  - **Comments**
  - **Processing Instructions (PI)**
  - **CDATA Sections**

XML – Markups and Content

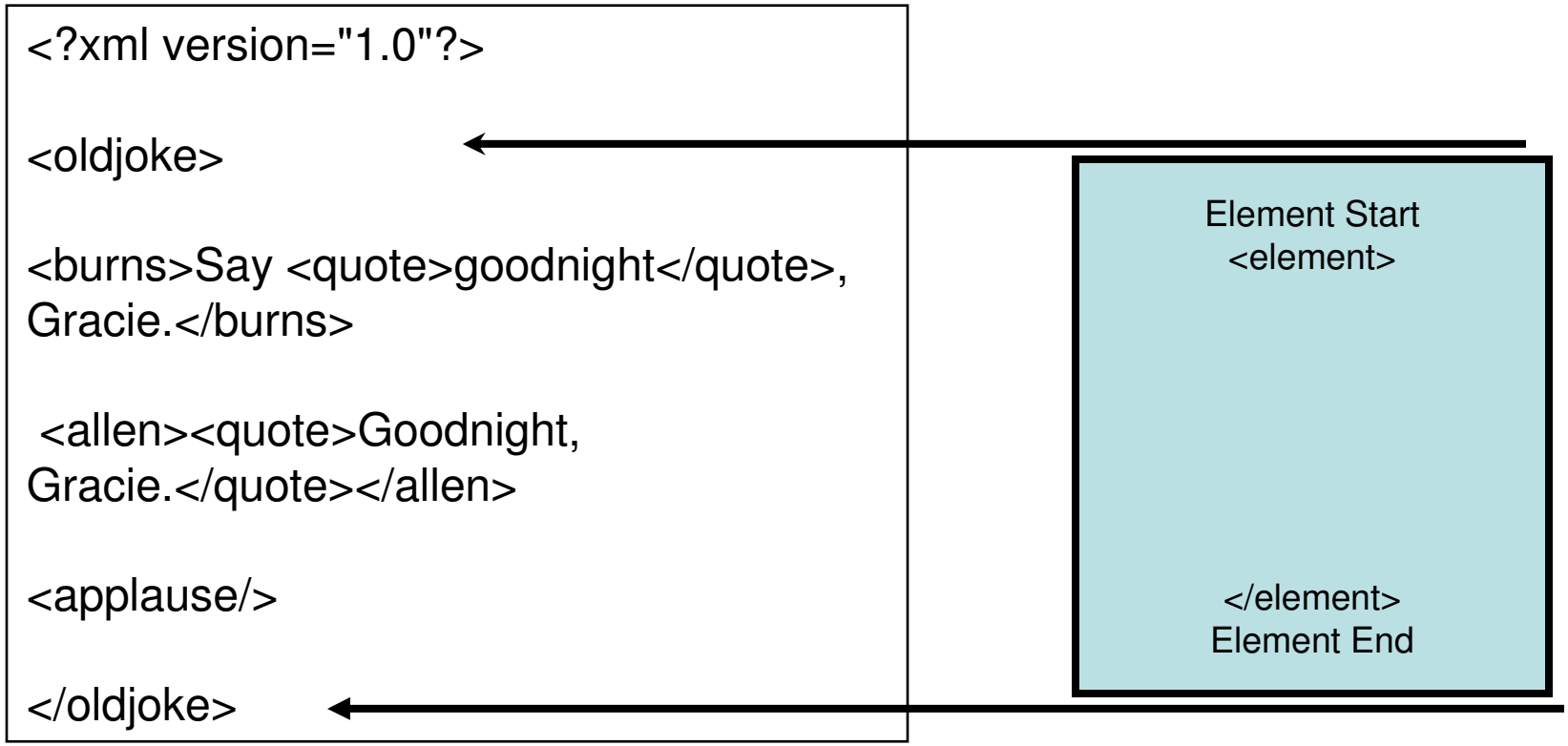
- **Elements**

Most common form of Markup

Delimited by angle brackets, generally named to identify the nature of the content they surround.

start-tag	***<element>***
end-tag	***</element>***
emtpy element	***<element/>***

What Do XML Documents Look Like? Element



What Do XML Documents Look Like? Element (Empty)

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

<oldjoke>

<burns>Say <quote>goodnight</quote>,
Gracie.</burns>

 <allen><quote>Goodnight,
Gracie.</quote></allen>

<applause/>

</oldjoke>
```

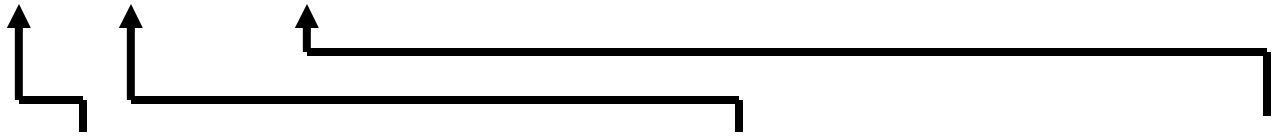
Empty Element, please note it would also be valid syntax to handle this in the following form: <applause></applause> as this is equivalent to <applause/>

XML – Markups and Content

– Attributes

Attributes are name-value pairs that occur **inside** start-tags after the element name. For example,

* *



is a **div** element with the attribute **class** having the value **preface**.

In XML, all attribute *values* must be quoted.

XML – Markups and Content

- Entity References

In order to introduce markup into a document, some characters have been reserved to identify the start of markup. The left angle bracket, < , for instance, identifies the beginning of an element start- or end-tag. In order to insert these characters into your document as content, there must be an alternative way to represent them.

In XML, entities are used to represent these special characters. Every Entity must have a unique name Entities are also used to refer to often repeated or varying text and to include the content of external files.

XML – Markups and Content

- Entity References (cont'd)

In order to use an entity, you simply reference it by name. Entity references begin with the ampersand and end with a semicolon.

For example, the lt entity inserts a literal < into a document.

So the string:

<element>

can be represented in an XML document as:

<element>

XML – Markups and Content

– Comments

Comments begin with <!-- and end with -->.
Comments can contain any data except the literal string --.
You can place comments between markup anywhere in your document.

Example:

<!-- This is my comment -->

Comments are not part of the textual content of an XML document.
An XML processor is not required to pass them along to an application.

XML – Markups and Content

- Processing Instructions (PI)

Processing instructions (PIs) are an escape hatch to provide information to an application. Like comments, they are not textually part of the XML document, but the XML processor is required to pass them to an application.

Processing instructions have the form: <?name pidata?>.

The name, called the PI target, identifies the PI to the application.

Applications will only process the targets they recognize and ignore all other PIs. Any data that follows the PI target is optional, it is for the application that recognizes the target. The names used in PIs may be declared as notations in order to formally identify them.

PI names beginning with xml are reserved for XML standardization.

XML – Markups and Content

– CDATA Sections

In a document, a CDATA section instructs the parser to ignore most markup characters.
Consider a source code listing in an XML document. It might contain characters that the XML parser would ordinarily recognize as markup (< and &, for example). In order to prevent this, a CDATA section can be used.

```
<![CDATA[

*p = &q;
b = (i <= 3);

]]>
```

Note: The only string that cannot occur in a CDATA section is]]>.

Between the start of the section, <![CDATA[and the end of the section,]]>, all character data is passed directly to the application, without interpretation. Elements, entity references, comments, and processing instructions are all unrecognized and the characters that comprise them are passed literally to the application.

XML Declarations

There are four kinds of declarations in XML:

- **element type**
- **attribute list**
- **entity**
- **notation**

XML Declarations

- **element type**

<!ELEMENT oldjoke (burns+, allen, applause?)>

This declaration identifies the element named oldjoke. Its *content model* follows the element name. The content model defines what an element may contain. In this case, an oldjoke must contain burns and allen and may contain applause. The commas between element names indicate that they must occur in succession. The plus after burns indicates that it may be repeated more than once but must occur at least once. The question mark after applause indicates that it is optional (it may be absent, or it may occur exactly once). A name with no punctuation, such as allen, must occur exactly once.

XML Declarations

- **element type (cont'd)**

<!ELEMENT oldjoke (burns+, allen, applause?)>

Declarations for burns, allen, applause and all other elements used in any content model must also be present for an XML processor to check the validity of a document

In addition to element names, the special symbol #PCDATA is reserved to indicate character data. The moniker PCDATA stands for parseable character data.

PCDATA Example:

<!ELEMENT burns (#PCDATA | quote)>*

XML Declarations

- **element type (cont'd)**

Elements that contain only other elements are said to have **element content.**

Elements that contain both other elements and #PCDATA are said to have **mixed content**.

Two other content models are possible: **EMPTY** indicates that the element has no content (and consequently no end-tag), and **ANY** indicates that *any* content is allowed. The **ANY** content model is sometimes useful during document conversion, but should be avoided because it disables all content checking in that element.

XML Declarations

Element Declarations – continuing example

<!ELEMENT oldjoke (burns+, allen, applause?)>

<!ELEMENT burns   (#PCDATA | quote)*>

<!ELEMENT allen   (#PCDATA | quote)*>

<!ELEMENT quote   (#PCDATA)*>

<!ELEMENT applause EMPTY>

XML Declarations

- **attribute list**

Attribute list declarations identify which elements may have attributes, what attributes they may have, what values the attributes may hold, and what value is the default. A typical attribute list declaration looks like this:

```
<!ATTLIST  oldjoke
name
ID
#REQUIRED

    label
CDATA
#IMPLIED

    status ( funny | notfunny ) 'funny'>
```

XML Declarations

-attribute list

```
<!ATTLIST  oldjoke
name
ID
#REQUIRED

    label
CDATA
#IMPLIED

    status ( funny | notfunny ) 'funny'>
```

In this example, the oldjoke element has three attributes:

name, which is an ID and is required;

label, which is a string (character data) and is not required;

status, which must be either funny or notfunny and defaults to funny, if no value is specified.

XML Declarations

-**attribute list**

Each attribute in a declaration has three parts:

- Name
- Type
- Default value

XML Declarations

- attribute list, Name:

For the most part you can select any name you wish but names cannot be repeated on the same element.

XML Declarations

-attribute list, Types:

- CDATA
  - CDATA attributes are strings, any text is allowed. Don't confuse CDATA attributes with CDATA sections, they are unrelated.
- ID
  - The value of an ID attribute must be a name. All of the ID values used in a document must be different. IDs uniquely identify individual elements in a document. Elements can have only a single ID attribute.
- IDREF or IDREFS
  - An IDREF attribute's value must be the value of a single ID attribute on some element in the document. The value of an IDREFS attribute may contain multiple IDREF values separated by white space.
- ENTITY or ENTITIES
  - An ENTITY attribute's value must be the name of a single entity. The value of an ENTITIES attribute may contain multiple entity names separated by white space.
- NMTOKEN or NMTOKENS
  - Name token attributes are a restricted form of string attribute. In general, an NMTOKEN attribute must consist of a single word but there are no additional constraints on the word, it doesn't have to match another attribute or declaration. The value of an NMTOKENS attribute may contain multiple NMTOKEN values separated by white space.

XML Declarations

-attribute list, Values:

There are four possible default values:

- #REQUIRED
  - The attribute must have an explicitly specified value on every occurrence of the element in the document.
- #IMPLIED
  - The attribute value is not required, and no default value is provided. If a value is not specified, the XML processor must proceed without one.
- "value"
  - An attribute can be given any legal value as a default. The attribute value is not required on each element in the document, and if it is not present, it will appear to be the specified default.
- #FIXED “value”
  - An attribute declaration may specify that an attribute has a fixed value. In this case, the attribute is not required, but if it occurs, it must have the specified value. If it is not present, it will appear to be the specified default.

XML Declarations

- entity

Entity declarations allow you to associate a name with some other fragment of content. That construct can be a chunk of regular text, a chunk of the document type declaration, or a reference to an external file containing either text or binary data.

Example Entity Declarations

<!ENTITY
ATI
"ArborText, Inc.">

<!ENTITY boilerplate SYSTEM
"/standard/legalnotice.xml">

<!ENTITY ATIlogo SYSTEM
"/standard/logo.gif" NDATA GIF87A>

XML Declarations

- entity

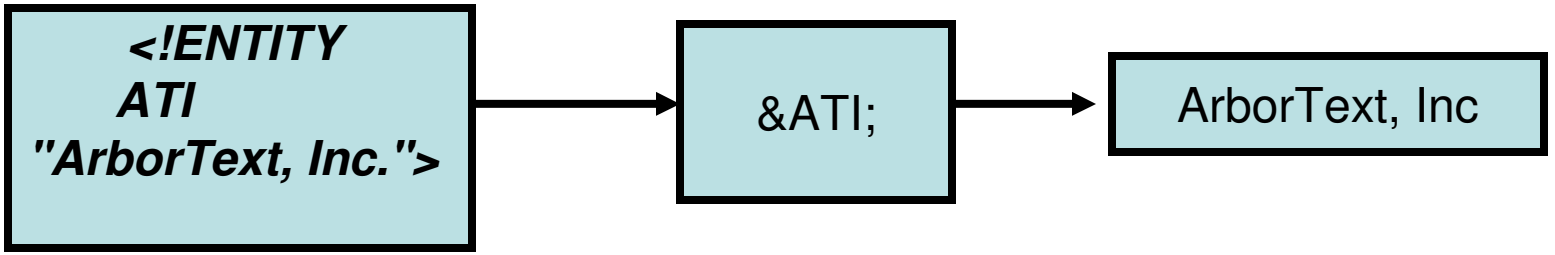
3 different kinds of entities:

- Internal
- External
- Parameters

XML Declarations

- entity, internal

Internal entities associate a name with a string of literal text. The first entity in the prior example set is an internal entity. Using &ATI; anywhere in the document will insert ArborText, Inc. at that location. Internal entities allow you to define shortcuts for frequently typed text or text that is expected to change, such as the revision status of a document.



XML Declarations

- entity, internal

The XML specification pre-defines five internal entities:

- < produces the left angle bracket, <
- > produces the right angle bracket, >
- & produces the ampersand, &
- ' produces a single quote character (an apostrophe), '
- " produces a double quote character, "

XML Declarations

- entity, external

- External entities associate a name with the content of another file. External entities allow an XML document to refer to the contents of another file. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and parsed as part of the referring document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document.

- The second and third entities in the prior example are external entities.

- Using &boilerplate; will have insert the *contents* of the file /standard/legalnotice.xml at the location of the entity reference. The XML processor will parse the content of that file as if it occurred literally at that location.

- The entity ATIlogo is also an external entity, but its content is binary. The ATIlogo entity can only be used as the value of an ENTITY (or ENTITIES) attribute (on a graphic element, perhaps). The XML processor will pass this information along to an application, but it does not attempt to process the content of /standard/logo.gif.

XML Declarations

- entity, parameter

Parameter entities can only occur in the document type declaration. A parameter entity declaration is identified by placing % (percent-space) in front of its name in the declaration. The percent sign is also used in references to parameter entities, instead of the ampersand. Parameter entity references are immediately expanded in the document type declaration and their replacement text is part of the declaration, whereas normal entity references are not expanded.

Parameter entities are not recognized in the body of a document.

XML Declarations

- entity, parameter

In use:

Looking back at the element declarations example, two of them have the same content model:

<!ELEMENT burns (#PCDATA | quote)>*
<!ELEMENT allen (#PCDATA | quote)>*

At the moment, these two elements are the same only because they happen to have the same literal definition. In order to make more explicit the fact that these two elements are semantically the same, use a parameter entity to define their content model. The advantage of using a parameter entity is two-fold. First, it allows you to give a descriptive name to the content, and second it allows you to change the content model in only a single place, if you wish to update the element declarations, assuring that they always stay the same:

```
<!ENTITY % personcontent "#PCDATA | quote">
<!ELEMENT burns (%personcontent;)*>
<!ELEMENT allen (%personcontent;)*>
```

XML Declarations

-Notation

Notation declarations identify specific types of external binary data. This information is passed to the processing application, which may make whatever use of it it wishes.

Example:

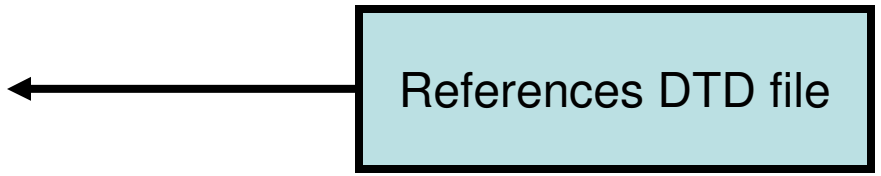
<!NOTATION GIF87A SYSTEM "GIF">

DTD (Document Type Declaration)

- In applications where a person composes or edits the data (as opposed to data that may be generated directly from a database, for example), a DTD is probably going to be required if any structure is to be guaranteed.
- If present, the document type declaration must be the first thing in the document

Example:

```
<?XML version="1.0" standalone="no"?>
<!DOCTYPE chapter SYSTEM "dbook.dtd" [
<!ENTITY %ulink.module "IGNORE">
<!ELEMENT ulink (#PCDATA)*>
<!ATTLIST ulink
    xml:link       CDATA  #FIXED "SIMPLE"
    xml-attributes CDATA  #FIXED "HREF URL"
    URL            CDATA  #REQUIRED>
]>
<chapter>...</chapter>
```



Validity

Given the preceding discussion of type declarations, it follows that some documents are valid and some are not.

There are two categories of XML documents: well-formed and valid.

Validity – Well-formed

A document can only be well-formed if it obeys the syntax of XML. A document that includes sequences of markup characters that cannot be parsed or are invalid cannot be well-formed.

In addition, the document must meet all of the following conditions:

- The document instance must conform to the grammar of XML documents. In particular, some markup constructs (parameter entity references, for example) are only allowed in specific places. The document is not well-formed if they occur elsewhere, even if the document is well-formed in all other ways.
- The replacement text for all parameter entities referenced inside a markup declaration consists of zero or more complete markup declarations. (No parameter entity used in the document may consist of only part of a markup declaration.)
- No attribute may appear more than once on the same start-tag.
- String attribute values cannot contain references to external entities.

Validity – Well-formed

In addition, the document must meet all of the following conditions: (cont'd)

- Non-empty tags must be properly nested.
- Parameter entities must be declared before they are used.
- All entities except the following: amp, lt, gt, apos, and quot must be declared.
- A binary entity cannot be referenced in the flow of content, it can only be used in an attribute declared as ENTITY or ENTITIES.
- Neither text nor parameter entities are allowed to be recursive, directly or indirectly.

By definition, if a document is not well-formed, it is not XML. This means that there is no such thing as an XML document which is not well-formed, and XML processors are not required to do anything with such documents.

Validity - Valid

A well-formed document is valid only if it contains a proper document type declaration and if the document obeys the constraints of that declaration (element sequence and nesting is valid, required attributes are provided, attribute values are of the correct type, etc.).

References

- Walsh, Norman. “A Technical Introduction to XML” Revision 1.1.18 Feb 1998, Date of access May 2005, http://xml.com/pub/a/98/10/guide0.html