XML 1.0

Péter Jeszenszky
Faculty of Informatics, University of Debrecen
jeszenszky.peter@inf.unideb.hu

Last modified: September 3, 2023

Standard

- The current standard:
  - *Extensible Markup Language (XML) 1.0 (Fifth Edition)* (W3C Recommendation 26 November 2008) https://www.w3.org/TR/xml/
- Describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them.

XML Documents

- Textual objects that are well-formed according to the standard.
- Each XML document has both a logical and a physical structure.
  - Physically, a document is composed of storage units called entities.
  - Logically, a document is composed of declarations, elements, comments, processing instructions, and other structural components.

Elements (1)

- Each element is delimited by a start-tag and an end-tag, or is is made up from a single empty-element tag.
  - Examples:

```
<author>Sir Arthur Conan Doyle</author>

<message xml:lang="en">Hello, World!</message>

<img src="logo.png" alt="Logo"/>
```

Elements (2)

- The name specified in a start-tag, end-tag, or empty-element tag is called the element type.
  - **Well-formedness constraint**: the name in an element's end-tag must match the element type in the start-tag.
- The start-tag and the end-tag surrounds the content of the element.
- Elements may have a set of name-value pairs called attribute specifications.

Elements (3)

- An element with no content is said to be empty.
  - The representation of an empty element is either a start-tag immediately followed by an end-tag, or an empty-element tag.
    - Example: `<elem></elem>`, `<elem/>`

Well-Formedness

- Roughly, it means the following:
  - A single top-level element called the **root element** contains all the other elements.
  - Each start-tag has a corresponding end-tag.
  - Elements are properly nested and and do not overlap.
  - Each entity referenced in the document is well-formed.
- Many other constrains, the so-called **well-formedness constraint**s, must be fulfilled.

Characters

- XML documents are composed of Unicode characters.
- All XML processors must accept the UTF-8 and UTF-16 encodings of Unicode.

Whitespace Characters (1)

- Whitespace characters are the following:
  - U+0009 (horizontal tabulation, HT, `'\t'`)
  - U+000A (line feed, LF, `'\n'`)
  - U+000D (carriage return, CR, `'\r'`)
  - U+0020 (space)

Whitespace Characters (2)

- In editing XML documents, it is often convenient to use whitespace that clearly shows the nesting of the elements to the human eye.
  - Placing an appropriate number of tabs or spaces at the beginning of lines.
- In general, these whitespace characters are not significant for computer programs.

Whitespace Characters (3)

- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<movie id="0078748"><title xml:lang="en">Alien</title>
<year>1979</year><genres><genre>horror</genre>
<genre>sci-fi</genre></genres></movie>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<movie id="0078748">
  <title xml:lang="en">Alien</title>
  <year>1979</year>
  <genres>
    <genre>horror</genre>
    <genre>sci-fi</genre>
  </genres>
</movie>
```

Name Characters

- See the specification for the list of characters allowed in names.
  - Letters and decimal digit characters are allowed.
  - Beside the usual `'_'` character, the `'.'`, `'-'`, `':'`, and `'·'` (U+00B7) characters are also available.
- **Name token** (**Nmtoken**): a sequence of one or more name characters.

Names (1)

- A name is a name token with a restricted set of initial characters.
  - See the specification for the list of allowed initial characters.
  - For example, names must not start with a digit, or any of the following characters: `'-'`, `'.'`, and `'·'` (U+00B7).
- The *Namespaces in XML Recommendation* assigns a special meaning to the `':'` character in names.
  - Therefore, the `':'` character should not be used except for namespace purposes.

Names (2)

- For example, the following character sequences are legal names:
  - `date.of.birth`
  - `date-of-birth`
  - `jaxb:version`
  - `item1`, item2, item3, …
  - `β0`
  - `_` (yes, a single underscore itself!)
  - `_-_-_` (yes, even this one!)
- For example, the following character sequences are not legal names:
  - `1st`, `2nd`, `3rd`, … (because a name must not start with a digit)
  - `on/off` (because `'/'` characters are not allowed in names)

Names (3)

- Names in XML (e.g., element and attribute names) are case-sensitive.
  - For example, `title`, `Title`, and `TITLE` are different names.

Literals

- Character sequences enclosed between `'"'` or `"'"` characters that does not contain the delimiter.
  - Example: `"/style.css"`, `'/style.css'`
- Used for, e.g., specifying the following:
  - The values of attributes
  - The content of internal entities

Markup

- Start-tag, end-tag, and empty-element tag
- Character reference
- Entity reference
- Comment
- Processing instruction
- CDATA section delimiters
- XML declaration
- Text declaration
- Document type declaration

Character Data

- All text that is not markup constitutes the character data of the document.

Language Identification (1)

- A special attribute named `xml:lang` may be inserted in documents to specify the language used in the contents and attribute values of any element.
  - In valid documents, this attribute, like any other, must be declared if it is used.
  - The attribute applies to the element where it is specified (including the values of its attributes), and to all elements in its content unless overridden with another instance of `xml:lang`.

Language Identification (2)

- The values of the attribute are language identifiers as defined by the following specification:
  - Addison Phillips (ed.), Mark Davis (ed.). *RFC 5646: Tags for Identifying Languages.* September 2009. https://www.rfc-editor.org/rfc/rfc5646
- Legal attribute values include: `de-AT`, `en-US`, `hu`, `az-Arab`, `az-Cyrl`, `az-Latn`, ...
- Specially, the empty value specifies that there is no language information available.
  - Just as if `xml:lang` had not been specified on the element or any of its ancestors.

Language Identification (3)

- Example:
  - `<para xml:lang="en">The more <phrase xml:lang="fr">outré</phrase> and grotesque an incident is the more carefully it deserves to be examined, and the very point which appears to complicate a case is, when duly considered and scientifically handled, the one which is most likely to elucidate it.</para>`
    - Arthur Conan Doyle, *The Hound of the Baskervilles*.

Special Characters

- The `'&'` and the `'<'` characters must not appear in their literal form, except when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section.
  - If they are needed elsewhere, they must be specified using either character references or the `&` and `<` entity references.
  - The `'>'` character may be represented using the `>` entity reference.
- To allow attribute values to contain both single and double quotes, the `'''` and `'"'` characters may be specified as `&apos;` and `&quot;`, respectively.

Start-Tags, End-Tags, and Empty-Element Tags (1)

- **Start-tag**:
  - E.g.: `<title>`, `<title xml:lang="en">`
- **End-tag**:
  - E.g.: `</title>`
- **Empty element tag**:
  - E.g.: `<br/>`, `<hr />`, `<img src="logo.png" alt="Logo"/>`

Start-Tags, End-Tags, and Empty-Element Tags (2)

- **Well-formedness constraint**: an attribute name must not appear more than once in the same start-tag or empty-element tag.
- The order of attribute specifications in a start-tag or empty-element tag is not significant.

Character References

- In text, attribute values, and literal entity values Unicode characters may also be expressed using character references of the form:
  - &#*nnnn*; where *nnnn* is a sequence of decimal digits representing the code point.
    - Examples: `&#169;` (©), `&#9775;` (☯), `&#128570;`
  - &#x*hhhh*; where *hhhh* is a sequence of hexadecimal digits representing the code point.
    - Examples: `&#xA9;` (©), `&#x262F;` (☯), `&#x1F63A;`

Entity References

- An entity reference refers to the content of a named entity.
  - Reference to a parsed general entity: *&name;*
    - Examples: `&`, `&Aacute;`, `&copyright;`
  - Parameter-entity reference: *%name;*
    - Examples: `%inline;`, `%ImgAlign;`
- Entity references are discussed later with the physical structures.

Comments

- They may appear anywhere in a document outside other markup.
  - The only exception is the document type declaration, in which they may appear at some places.
- Example:
  - `<!-- This is a comment -->`

Processing Instructions

- They contain instructions for applications.
- Example:
  - `<?xml-stylesheet type="text/css" href="style.css"?>`

CDATA Sections

- They may occur anywhere where character data may occur.
  - They are used to escape blocks of text containing characters which would otherwise be recognized as markup.
  - Within a CDATA section, only the `']]>'` string is recognized as markup.
- Example:
  - `<![CDATA[if (0 < n && n <= 10)]]>`

XML Declaration

- XML documents should begin with an XML declaration which specifies the version of XML being used.
  - The character encoding must be specified if the encoding used is not UTF-8 or UTF-16, unless an encoding is determined by a higher-level protocol (e.g., HTTP).
- Examples:
  - `<?xml version="1.0"?>`
  - `<?xml version='1.0' encoding='UTF-8'?>`

Document Type Declaration (1)

- It contains and/or points to markup declarations that provide a grammar for a class of documents.
  - This grammar is known as a **document type definition**, or DTD.
- The name in the document type declaration specifies the element type of the root element.

Document Type Declaration (2)

- Document type declarations that point to an external entity containing markup declarations called the **external DTD subset**:
  - `<!DOCTYPE score-partwise SYSTEM "http://www.musicxml.org/dtds/partwise.dtd">`
  - `<!DOCTYPE score-partwise SYSTEM "partwise.dtd">`
  - `<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN" "http://www.musicxml.org/dtds/partwise.dtd">`
- The above examples use the MusicXML DTD.
  - See: https://www.musicxml.com/

Document Type Declaration (3)

- A document type declaration that contains markup declarations called the **internal DTD subset**:
  -
```
<!DOCTYPE message [
   <!ELEMENT message (#PCDATA)>
   <!ATTLIST message
      xml:lang CDATA #IMPLIED>
]>
```

Document Type Declaration (4)

- A document type declaration that points to an external DTD subset and also contains an internal DTD subset:
  -
```
<!DOCTYPE play SYSTEM "play.dtd" [
  <!ENTITY r "Rosencrantz">
  <!ENTITY g "Guildenstern">
  <!ENTITY rag "&r; and &g;">
]>
```

Document Type Definition (1)

- It allows to define constraints on the logical structure of documents and supports the use of storage units.
- It consists of markup declarations.

Document Type Definition (2)

- It can be associated with a document in the document type declaration.
  - The document type declaration can point to an external DTD subset, can contain an internal DTD subset, or can do both.
- The DTD for a document consists of both subsets taken together.
  - If both the external and internal subsets are present, the internal subset must be considered to occur before the external subset.
    - This allows the overriding of entity and attribute-list declarations from the external subset.

Validity (1)

- An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it.
  - The specification defines the constraints that must be fulfilled. These are the so-called **validity constraint**s.
  - Any violation of the validity constraints is an error.

Validity (2)

- Roughly, it means the following:
  - The name in the document type declaration must match the element type of the root element.
  - Each element in the document must be declared in the DTD and its content must conform to the declaration.
  - Each attribute in the document must be declared in the DTD and its value must conform to the declaration.
  - If an attribute is declared as required in the DTD, it must be specified explicitly in the document for each occurrence of the element type.

Markup Declarations

- Element type declaration
- Attribute-list declaration
- Entity declaration

Element Type Declaration

- An element type declaration constrains the element's content.
- **Validity constraint**: an element type must not be declared more than once.

Element Type Declaration: Empty Elements

- **Validity constraint**: such declared elements must have no content.
- Example for declaring and using an empty element:

```
<!ELEMENT br EMPTY>
…
<br/>
<br></br>
```

Element Type Declaration: Element Content (1)

- An element type has element content when elements of that type must contain only child elements (no character data), optionally separated by whitespace.
  - Comments and processing instructions are also allowed between child elements.
- The declaration specifies a **content model**, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.
  - The content model is a regular expression-like pattern.

Element Type Declaration: Element Content (2)

- Constructs available for specifying content models:
  - Sequence list, such as
    ```
    (street, city, zip, country)
    ```
  - Choice list, such as
    ```
    (ul | ol | dl)
    ```
  - Special characters governing the number of occurrences (they apply to the preceding subexpression):
    - **?**: zero or one (0, 1)
    - **+**: one or more (1, 2, 3, …)
    - *****: zero or more (0, 1, 2, …)
- Complex expressions can be built from these constructs.

Element Type Declaration: Element Content (3)

- Assume that the elements `f`, g, and h are declared to be empty.

Declaration	The content of e	Valid instances of e
`<!ELEMENT e (f)>`	A single `f` element	`<e><f/></e>`
`<!ELEMENT e (f?)>`	0 or 1 `f` element	`<e></e>` `<e><f/></e>`

`<!ELEMENT e (f+)>`	1 or more `f` elements	`<e><f/></e>`
		`<e><f/><f/></e>`
		`<e><f/><f/><f/></e>`
		…

`<!ELEMENT e (f*)>`	0 or more `f` elements	`<e></e>`
		`<e><f/></e>`
		`<e><f/><f/></e>`
		…

`<!ELEMENT e (f,g,h)>`	An `f`, a g, and an h element, in this order	`<e><f/><g/><h/></e>`

`<!ELEMENT e (f|g|h)>`	Either a single `f`, g, or h element	`<e><f/></e>`
		`<e><g/></e>`
		`<e><h/></e>`

Element Type Declaration: Element Content (4)

- **Example**: declaring an element a that must contain an even number (0, 2, 4, …) of b elements:
  - `<!ELEMENT a (b,b)*>`
- **Example**: declaring an element a that must contain an odd number (1, 3, 5, …) of b elements:
  - `<!ELEMENT a (b,(b,b)*)>`
- **Example**: declaring an element a that must contain 1, 2, or 3 b elements:
  - `<!ELEMENT a (b,(b,b?)?)>`

Element Type Declaration: Element Content (5)

- **Validity constraint**: the content of such declared elements must conform to the content model in the declaration.
  - Whitepace, comments and processing instructions are allowed before the first child element, between child elements, and after the last child element.

Element Type Declaration: Mixed Content (1)

- An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements.
  - The types of the child elements may be constrained, but not their order or their number of occurrences.

Element Type Declaration: Mixed Content (2)

- Example for declaring and using an element with mixed content:

```
<!ELEMENT para (#PCDATA | link)*>
<!ELEMENT link (#PCDATA)>
…
<para>See
  <link>https://sass-lang.com/</link>
  and
  <link>https://stylus-lang.com/</link>.
</para>
```

Element Type Declaration: Mixed Content (3)

- **Validity constraint**: after replacing any entity references with their replacement text, such declared elements must contain character data, CDATA sections, comments, processing instructions, and child elements whose types match names in the content model.
- **Validity constraint**: the same name must not appear more than once in a mixed-content declaration.

Element Type Declaration: Text-Only Element Content (1)

- A special case of mixed content declaration is used to declare a text-only element.
- Example:

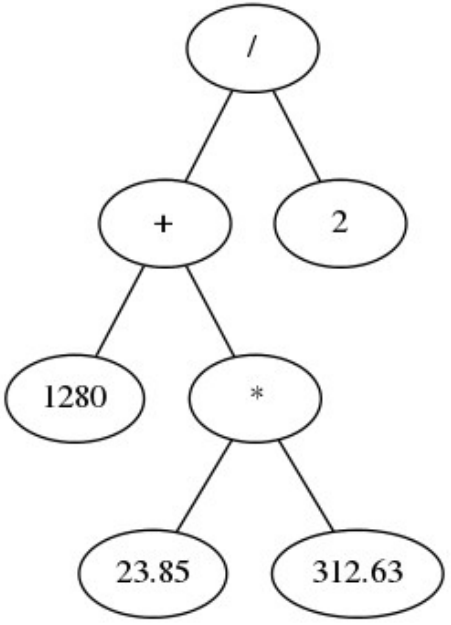
```
<!ELEMENT title (#PCDATA)>
…
<title>Angels &amp;amp; Demons</title>
```

Element Type Declaration: Text-Only Element Content (2)

- **Validity constraint**: after replacing any entity references with their replacement text, such declared elements must contain character data (including CDATA sections), comments, and processing instructions.

Element Type Declaration: Practical Example (1)

- Content models can be used to describe document structures that are deeply nested.
- Example: binary expression tree
  - The binary expression tree representing the arithmetic expression (1280 + (23.85 * 312.63)) / 2:



Element Type Declaration: Practical Example (2)

- `expression.dtd`:

```
<!ELEMENT node ((node | number), (node | number))>
<!ATTLIST node
  op CDATA #REQUIRED>

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

Element Type Declaration: Practical Example (3)

- `expression.xml`:

```
<?xml version="1.0"?>
<!DOCTYPE node SYSTEM "expression.dtd">
<node op="/">
    <node op="+">
        <number>1280</number>
        <node op="*">
            <number>23.85</number>
            <number>312.63</number>
        </node>
    </node>
    <number>2</number>
</node>
```

Attribute-List Declarations (1)

- They specify the name, data type, and default value (if any) of each attribute associated with a given element type.
- More than one can be provided for a given element type, the contents of all those provided are merged.
- When more than one definition is provided for the same attribute of a given element type, the first declaration is used and later declarations are ignored.
- It is not an error to declare attributes for an element type that is not itself declared.

Attribute-List Declarations (2)

- They have the following syntax:

  `<!ATTLIST` *name*

  (*name type default_value*)*>

  - The first name in the declaration is the type of an element.
  - The name is followed by triplets each of which declares an attribute for the given element type.
    - The triplets consist of an attribute name, an attribute type, and a default value declaration.

Attribute-List Declarations (3)

- Examples:

```
<!ATTLIST item id ID #REQUIRED>

<!ATTLIST property
   name CDATA #REQUIRED
   value CDATA #REQUIRED>

<!ATTLIST answer correct (yes|no) "no">

<!ATTLIST todoList
   xmlns CDATA #FIXED
     "http://www.example.com/todoList">
```

Attribute Types

- Attribute types are of three kinds:
  - **String type** (`CDATA`)
  - **Tokenized types** (`ID`, `IDREF`, `IDREFS`, `NMTOKEN`, `NMTOKENS`)
  - **Enumerated types** (enumerations)
- Attributes of type `CDATA` may take any literal string as a value.
- The other types impose validity constraints on attribute values.
  - These validity constraints are applied after the attribute value has been normalized.

Attribute Types: ID

- **Validity constraint**: attribute values of type `ID` must be names that must not appear more than once in an XML document as an attribute value of this type.
  - Thus, `ID` values must uniquely identify the elements which bear them.
- **Validity constraint**: an element type must not have more than one attribute of type `ID` specified.
- **Validity constraint**: an attribute of type `ID` must have a declared default of `#IMPLIED` or `#REQUIRED`.

Attribute Types: IDREF, IDREFS

- **Validity constraint**: attribute values of type `IDREF` must be names that match the value of an attribute of type `ID` on some element in the XML document.
- **Validity constraint**: attribute values of type `IDREFS` must be space-separated lists of one or more names, where each name matches the value of an attribute of type `ID` on some element in the XML document.

Attribute Types: NMTOKEN, NMTOKENS

- **Validity constraint**: attribute values of type `NMTOKEN` must be name tokens.
- **Validity constraint**: attribute values of type `NMTOKENS` must be space-separated lists of one or more name tokens.

Attribute Types: Enumerations

- Enumerated attributes have a list of allowed values in their declaration.
- **Validity constraint**: the name tokens in the declaration must all be distinct.
- **Validity constraint**: attribute values of this type must match one of the name tokens in the declaration.

Attribute Defaults (1)

- The default declaration provides information on whether the attribute's presence is required, and if not, how an XML processor is to react if a declared attribute is absent in a document.
- The following options are available:
  - `#REQUIRED`
    - Means that the attribute must always be provided.
  - `#IMPLIED`
    - Means that the attribute need not be provided and there is no default value.
  - *literal*
    - Means that the attribute need not be provided, and this is the default in the absence of the attribute.
  - `#FIXED` *literal*
    - Means that if specified, the attribute must always take this value, and this is default in the absence of the attribute.

Attribute Defaults (2)

- **Validity constraint**: if the default declaration is the keyword `#REQUIRED`, then the attribute must be specified for all elements of the type in the attribute-list declaration.
- **Validity constraint**: the declared default value must meet the syntactic constraints of the declared attribute type.
- **Validity constraint**: if an attribute has a default value declared with the `#FIXED` keyword, instances of that attribute must match the default value.

Entities (1)

- Physically, XML documents are made up of storage units called entities.
- The entity that serves as the starting point for the XML processor is called the **document entity**.
  - This may contain the whole document.
- They all have content.
- They are all identified by a name, except for the document entity and the external DTD subset.
  - Named entities are declared in the DTD.

Entities (2)

- Entities may be either parsed or unparsed.
- Unparsed entities are not important for us, so we will not discuss them.
- Therefore, in the following, all entities are parsed.

Parsed Entities

- A parsed entity contains text that is referred to as its replacement text.
  - The text may represent markup and/or character data.
- They are used for text substitution.
- They are invoked by name using entity references.
  - Processing an entity reference to a parsed entity results in the substitution of its replacement text in place of the reference itself.

Types of Entities (1)

- Each entity is either a general entity or a parameter entity.
  - General entities are for use within the document content.
  - Parameter entities are for use within the DTD.
- These two types of entities use different forms of reference and these are treated differently.
- Furthermore, these two types of entities occupy different namespaces.
  - A parameter entity and a general entity with the same name are two distinct entities.

Types of Entities (2)

- Each entity is either an internal entity or an external entity.
  - The content of an internal entity is given in the declaration itself.
  - The content of an external entity is provided by an external resource.

Entity Declarations (1)

- Declaring an internal general entity:
  - Example:
    ```
    <!ENTITY unideb "University of Debrecen">
    ```
    - The entity can be referred to using the entity reference `&unideb;`.
- Declaring an internal parameter entity:
  - Example: `<!ENTITY % lists "ul | ol | dl">`
    - The entity can be referred to using the entity reference `%lists;`. The reference is recognized in the DTD only!

Entity Declarations (2)

- Declaring an external general entity:
  - Example: `<!ENTITY copyright SYSTEM "copyright.xml">`
    - The entity can be referred to using the entity reference `&copyright;`.
- Declaring an external parameter entity:
  - Example:

```
<!ENTITY % svg-animation.mod
  PUBLIC "-//W3C//ELEMENTS SVG 1.1 Animation//EN"
  "svg-animation.mod">
```

    - The entity can be referred to using the entity reference `%svg-animation.mod;`. The reference is recognized in the DTD only!

Entity Declarations (3)

- The same entity can be declared in more than one entity declaration
  - If the same entity is declared more than once, the first declaration is used and later declarations are ignored.

A Complex Example of Using Entities

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd" [
   <!ENTITY gc "Grumpy Cat">
   <!ENTITY fw SYSTEM "foreword.xml">
   <!ENTITY ch1 SYSTEM "chapter1.xml">
   <!ENTITY ch2 SYSTEM "chapter2.xml">
   <!ENTITY ch3 SYSTEM "chapter3.xml">
   <!ENTITY biblio SYSTEM "bibliography.xml">
]>
<book>
   <author>&gc;</author>
   <title>How to be Grumpy?</title>
   &fw;
   &ch1;
   &ch2;
   &ch3;
   &biblio;
</book>
```

Predefined Entities

- Predefined entities for special characters: `amp` (`'&'`), `lt` (`'<'`), `gt` (`'>'`), `apos` (`''`), `quot` (`'"'`).
- All XML processors must recognize these entities whether they are declared or not.

Well-Formed Entities (1)

- An internal or external general parsed entity is well-formed if its replacement text contains character data interspersed with elements, character references, general entity references, CDATA sections, processing instructions, and comments.
  - External general parsed entities may begin with a text declaration which specifies the character encoding used.

Well-Formed Entities (2)

- A consequence of well-formedness in general entities is that the logical and physical structures in an XML document are properly nested.
  - No start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

Some Examples of DTDs

- Checkstyle
  https://checkstyle.org/dtds/configuration_1_3.dtd
- MathML
  http://www.w3.org/Math/DTD/mathml3/mathml3.dtd
- MusicXML
  https://github.com/w3c/musicxml/blob/v4.0/schema/partwise.dtd
- SVG
  http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd

XML Processors

- An XML processor is a software module for reading XML documents and providing access to their content and structure.
  - An XML processor is doing its work on behalf of another module, called the application.
- XML processors fall into two classes: validating and non-validating.

XML Format Design Considerations (1)

- How to properly name elements and attributes?
  - Recommended reading:
    - Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009.
      - *Chapter 2: Meaningful Names* also applies to XML names to a large extent.

XML Format Design Considerations (2)

- How to properly name elements and attributes? (continued)
  - Some practical guidelines:
    - Intention-revealing names should be used.
    - The use of misleading names should be avoided.
    - Noise words that are too general (e.g., `Data`, `Info`, or `Content`) should be avoided when choosing names.
    - You should use pronounceable names.
    - Names should be chosen consistently.
      - Use a naming convention (e.g., lowerCamelCase) consistently.

XML Format Design Considerations (3)

- When to use attributes?
  - How to represent some information: should I use either element content or attributes?
    - An attribute name must not be specified more than once for an element.
      - Thus, pieces of data that can occur more than once can not be provided for the same element using an attribute.
    - Note that it is not possible to constrain the order of attributes.
      - Thus, when the order matters, use elements instead of attributes.

XML Format Design Considerations (4)

- When to use attributes? (continued)
  - Example:

```
<bookmarks title="Links of Personal Interest">

  <item url="https://www.brothers-brick.com/" title="The Brothers Brick">
    <tag>lego</tag>
    <tag>news</tag>
    <tag>reviews</tag>
  </item>

  <item url="https://thy-catafalque.hu/" title="Thy Catafalque">
    <tag>music</tag>
    <tag>ambient</tag>
    <tag>avantgarde</tag>
    <tag>exprimental</tag>
    <tag>metal</tag>
    <tag>band</tag>
  </item>

</bookmarks>
```

XML Format Design Considerations (5)

- Further recommended reading:
  - *Google XML Document Format Style Guide* https://google.github.io/styleguide/xmlstyle.html