Object-Oriented Programming

XML

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Data in an Executable Program

- Store in memory with *data structures*

- Containers: *std::vector*, *std::list*, *std::array*, c-array

- Objects: *struct*, *class*

- Database

External Data

- Must store *persistent* data externally, e.g., files

- Must exchange data with other programs

- Example: One program produces data (a *producer*), another program converts the data to a web page (a *consumer*)

- Need to store and exchange using a *data format*

Text Data Formats

- CSV
- JSON
- YAML
- XML

XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student name="gmarx">
        <problem>Very <ability>bright</ability>.
Tends to chew on <banned>gum</banned> in class</problem>
    </student>
    <student name="moe">
        <problem>Keeps <annoyance>bothering fellow
            students</annoyance></problem>
    </student>
    <!-- ... -->
</students>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student>
        <name>jdoe</name>
    </student>
    <student>
        <name>moe</name>
    </student>
    <!-- ... -->
</students>
```

- General-purpose interchange data format
- Represents *structured* and *semi-structured* data
- Very applicable to text documents, e.g., poetry and source code
- Wide variety of tools and approaches

XML Technologies

- XPath - Addressing language
- XSLT - Transformation language
- DTD, XML Schema, Relax NG - Schema languages, specify structure and content
- DOM, JDOM, XOM, SAX2, LINQ - Parsers for XML, APIs to build on

Views of XML

Viewpoint	Concept	Characteristics
data	*XML is a data format*	No mixed content Lots of attributes Often a memory dump of internal objects Often inflexible
document	*XML is a document format*	Mixed content Favors elements over attributes High-level view of mixed data Very flexible to additional elements and attributes

Simple List in XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <name>jdoe</name>
    <name>moe</name>
    <!-- ... -->
</students>
```

XML Declaration

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

- First thing in an XML document
- Optional, but always include
- *version* - Although there is an XML 1.1, most XML is 1.0.
- *encoding* - Text encoding
- UTF-8 is the default and the most common. UTF-8 is an extension of ASCII.
- *standalone* - Ignore any markup declarations in the DTD
- Except for exceptional situations, these are the only values you will see

Elements

Part	Example
Element	`<name>...</name>`
Element Start Tag	`<name>`
Element End Tag	`</name>`
Empty Element	`<name></name>` `<name/>`

XML Element

- XML is a markup language

- **There are no specific XML tag names defined in the XML standard**

- The specific XML element names (and attributes) are defined for each *application*, e.g., srcML elements

- Defining these names is not trivial

- Use standard sets of XML elements whenever possible, e.g., XHTML

Element Content

Content Type	Example	Value
Text	`<name>cin</name>`	"cin" is the text content of the element name
Nested elements	`<expr><name>a</name> + <name>b</name></expr>`	Element "name" is nested inside of element expr
Element		The *start tag*, the *end tag*, and all content in between

Nested Elements in XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student>
        <name>jdoe</name>
    </student>
    <student>
        <name>moe</name>
    </student>
    <!-- ... -->
</students>
```

Attributes

```
<comment type="block">
```

- Elements can have *attributes*, which are name-value pairs listed in the start tag
- E.g., *name* is "type"
- E.g., *value* is "block"
- Attributes must be unique to an element, i.e., multiple attributes with the same name are not allowed
- The value of an attribute is a string and cannot have nested elements

Attributes in XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student name="jdoe"/>
    <student name="moe"/>
    <!-- ... -->
</students>
```

Attributes vs. Subelements

	Example	Nested Text	Nested Elements	Purpose
Attributes	`<student name="jdoe"/>`	Yes	No	*metadata*
Subelements	`<student><name>jdoe</name></student>`	Yes	Yes	*data*

well-formed

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student name="gmarx">
        <problem>Very <ability>bright</ability>.
Tends to chew on <banned>gum</banned> in class</problem>
    </student>
    <student name="moe">
        <problem>Keeps <annoyance>bothering fellow
            students</annoyance></problem>
    </student>
    <!-- ... -->
</students>
```

- Requirement of XML

- Exactly one *root element*, `<students>...</students>`

- Elements form a tree structure and nest properly

- All elements are an empty element or have a *start tag* and an *end tag*

well-formed Violations

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student name="gmarx">
        <problem>Very <ability>bright</ability>.
Tends to chew on <banned>gum</banned> in class</problem>
    <student name="moe">
        <problem>Keeps <annoyance>bothering fellow
            students</annoyance></problem>
    </student>
    <!-- ... -->
</students>

<students xmlns="https://mlcollard.net/Student">
    <student name="harpo">
        <problem>Doesn't say anything</problem>
    </student>
</students>
```

- More than one root element
- Missing end tags
- Not nested properly
- XML processors are required to stop processing and report an error if a document is not well-formed

Escaped Characters

Unescaped	Escaped	Requirements
<	<	escape required
&	&	escape required
>	>	escape optional
'	'	escape depends on context
"	"	escape depends on context

Escaped Text

Unescaped	`#include <iostream>`
Escaped	`#include &lt;iostream&gt;`

Mixed Content

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<students xmlns="https://mlcollard.net/Student">
    <student name="gmarx">
        <problem>Very <ability>bright</ability>.
Tends to chew on <banned>gum</banned> in class</problem>
    </student>
    <student name="moe">
        <problem>Keeps <annoyance>bothering fellow
            students</annoyance></problem>
    </student>
    <!-- ... -->
</students>
```

- Element has both text and nested elements

- Additional markup for a section item

- Good for text with included markup

Namespaces

Namespace Declaration	`xmlns:s="https://mlcollard.net/Student"`
Prefix	`s`
URI	`https://mlcollard.net/Student`
Element in the namespace	`<s:grade>`
Default Namespace Declaration	`xmlns="https://mlcollard.net/Student"`

Namespaces with Default Prefix

```
<?xml version="1.0" encoding="UTF-8" standlone="yes" ?>
<students xmlns="https://mlcollard.net/Student">
    <student name="gmarx">
        <problem>Very <ability>bright</ability>.
Tends to chew on <banned>gum</banned> in class</problem>
    </student>
    <student name="moe">
        <problem>Keeps <annoyance>bothering fellow
            students</annoyance></problem>
    </student>
    <!-- ... -->
</students>
```

Namespaces with Prefix

```
<?xml version="1.0" encoding="UTF-8" standlone="yes" ?>
<s:students xmlns:s="https://mlcollard.net/Student">
    <s:student name="gmarx">
        <s:problem>Very <s:ability>bright</s:ability>.
            Tends to chew on <s:banned>gum</s:banned> in class</s:problem>
    </s:student>
    <s:student name="moe">
        <s:problem>Keeps <s:annoyance>bothering fellow
            students</s:annoyance></s:problem>
    </s:student>
    <!-- ... -->
</s:students>
```

Namespace Matching

```
<s:student xmlns:s="https://mlcollard.net/Student"><s:name>jdoe</s:name></s:student>

<student xmlns="https://mlcollard.net/Student"><name>jdoe</name></student>
```

- Element name matching is based on namespace URI, **not** on the prefix. Both of these are identical

- Many XML processors (tools that work on XML) require a non-default prefix for certain functionality, e.g., XPath in *libxml2*

Multiple Namespaces

```
<?xml version="1.0" encoding="UTF-8" standlone="yes" ?>
<s:students xmlns:s="https://mlcollard.net/Student"
            xmlns:c="https://mlcollard.net/Issues">
    <s:student name="gmarx">
        <s:problem>Very <c:ability>bright</c:ability>.
            Tends to chew on <c:banned>gum</c:banned> in class</s:problem>
    </s:student>
    <s:student name="moe">
        <s:problem>Keeps <c:annoyance>bothering fellow
            students</c:annoyance></s:problem>
    </s:student>
    <!-- ... -->
</s:students>
```

Whitespace

```
<student name="jdoe"/>
<student     name = "jdoe" />
```

- *significant* and *insignificant*
- Newlines are normalized (i.e., converted) to Unix line endings **even on Windows**
  - CR + LF → LF
- Whitespace in tags is insignificant and is just a separator, i.e., both start tags on the left are equivalent
- XML processors will often remove extra insignificant whitespace
- White space in element content can be significant. Assume that it is.

Validating for well-formedness

```
xmllint --noout data/demo.xml
```

- Any XML producer must produce well-formed XML, or the consumer will not process it

- `xmllint` - Tool for checking if XML is well-formed

- Ubuntu package: *libxml2-utils*

- Preinstalled on macOS and in our image for GitHub Codespaces

- The command on the left checks if XML is well-formed

Further Validation

- Allowed elements, text content of elements, and nested elements are specific to each XML application (i.e., each XML format)

- Must be defined in an XML grammar
  - *DTD*
  - *XSchema*
  - *RelaxNG*

Special XML Contents

- XML comment

- CDATA section

- XML Processing Instructions

XML Comment

XML Comment	`<!-- <s:student> -->`
XML Comment Start	`<!--`
XML Comment End	`-->`

XML Terminology

Term	Expansion	Example	Relation to Namespaces
qName	*qualified name*	`s:student`	Unique
prefix	*prefix*	`s`	Shared
localName	*local name*	`student`	May exist in multiple
URI	*Uniform Resource Identifier*	`https://mlcollard.net/Student`	URLs are a subset of URIs

XML Interface Concepts

Concept	Attributes	Notes
Start Document		Occurs before parsing
XML Declaration	*version*, *encoding*, *standalone*	Occurs once before the root
Element Start Tag	*qName*, *prefix*, *localName*	
Element End Tag	*qName*, *prefix*, *localName*	
Characters	*characters*	Includes entity references
Attribute	*qName*, *prefix*, *localName*, *value*	
XML Namespace	*prefix*, *uri*	
XML Comment	*value*	
CDATA	*characters*	Somewhat rare, and not in our data
Processing Instruction	*target*, *data*	Rare, and not in our data
End Document		Occurs after parsing