XML Processing

olsen software

Contents

Demo folder: **08-XML**

1. XML Essentials

- The need for XML
- XML document structure
- Defining elements
- Defining a hierarchy of elements
- Defining empty elements
- Defining attributes
- Defining XML namespaces

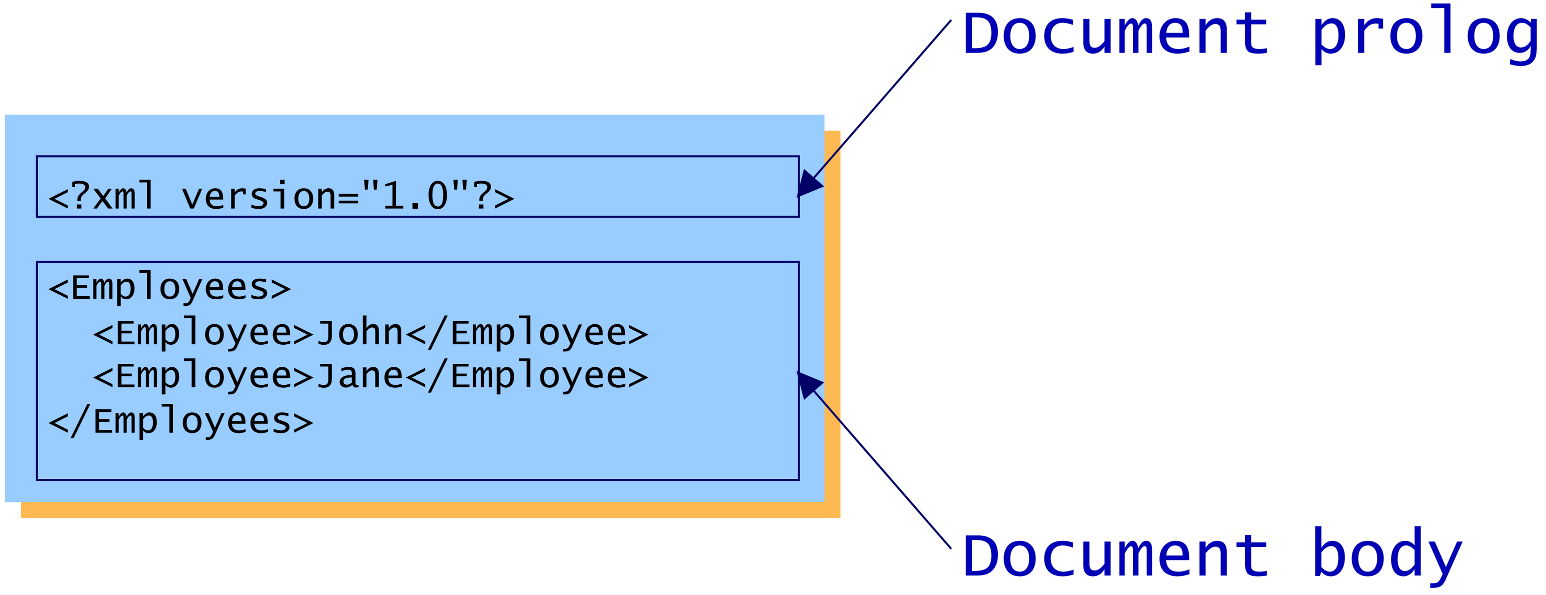
The Need for XML

- Companies and organizations face many difficulties in processing and exchanging data
  - Data integration
  - Document interchange
  - Document aggregation
  - Data representation

- The Internet era also raises technical difficulties as companies seek to integrate widely distributed solutions
  - RMI, DCOM, and CORBA are not Internet-friendly

- XML overcomes many of these problems

XML Document Structure



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

```
<Employees>
  <Employee>John</Employee>
  <Employee>Jane</Employee>
</Employees>
```

Defining Elements

- An element with text content

```
<Town>Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch</Town>
```

- White space in the text content is significant

```
<Greeting>  H  E  L  L  O   !!! </Greeting>
```

- XML is case-sensitive (unlike HTML)

```
<SpotTheBug>oops</SpotTheBUG>  ☠
```

Defining a Hierarchy of Elements

```xml
<FlightReservation>

  <PartyDetails>
    <NumberOfAdults>2</NumberOfAdults>
    <NumberOfChildren>2</NumberOfChildren>
  </PartyDetails>

  <ContactDetails>
    <Name>Angela Smith</Name>
    <Address>
      <HouseNumber>4</HouseNumber>
      <Postcode>SW1 1BA</Postcode>
    </Address>
  </ContactDetails>

  <BookingDetails>
    <BookingCode>FZ87YB62</BookingCode>
    <DateOfBooking>2018-01-31</DateOfBooking>
  </BookingDetails>

  <FlightDetails>
    <DateOfFlight>2018-02-28</DateOfFlight>
    <FlightCode>BA001</FlightCode>
  </FlightDetails>

</FlightReservation>
```

HierarchyExample.xml

Defining Empty Elements

- An empty element

  `<hr/>` equivalent to... `<hr></hr>`

- Empty elements are often used with attributes
  - See next slide ☺

- Empty elements are also often used in XSLT style sheets that translate XML into HTML

Defining Attributes

- You can define attributes in the start tag of an element
  - Attributes provide qualifying information to elements

```
<UnitPrice currency="GBP" salesTaxRate="17.5">34.95</UnitPrice>
```

- Attributes are often used in empty elements

```
<Address HouseNumber="4" PostCode="SW1 1BA"/>
```

- Attribute values must be enclosed in quotes
  - You can use "double quotes" or 'single quotes'

Defining XML Namespaces

- Namespaces enable you to qualify elements (usually) and attributes (sometimes), to avoid name clashes

- You define namespaces in an element start-tag:

```
<elem-start-tag  xmlns:namespace-prefix=namespace-uri  …>
```

- Simple example:

```
<ns:Employees xmlns:ns="http://www.mydomain.com">

  <ns:Employee>
    <ns:Name>John Smith</ns:Name>
    <ns:Salary>25000</ns:Salary>
  </ns:Employee>

  … etc. …

</ns:Employees>
```

NamespaceExample.xml

2. Reading XML Data in Python

- Overview of the ElementTree XML API
- Sample document
- Parsing XML
- Getting element information
- Iterating over child elements
- Indexing into child elements
- Finding specific child elements
- Iterating over descendant elements

Overview of the ElementTree XML API

- The `ElementTree` XML API is a standard and efficient Python library for parsing and creating XML data
  - Located in the `xml.etree` module

- The first step is to import `ElementTree`
  - It's conventional to alias it as `ET`, for brevity

```
import xml.etree.ElementTree as ET
```

- Note:
  - All the demo code for this section is in `readXml.py`

Sample Document

```xml
<?xml version="1.0"?>
<Company Name="Acme" YearOfIncorporation="1997">

  <Employee EmpNum="456" JobTitle="Programmer">
    <Name>
      <FirstName>Matthew</FirstName>
      <LastName>Williams</LastName>
    </Name>
    <Tel>222-7777-111</Tel>
    <Tel>222-7777-222</Tel>
    <Tel>222-7777-333</Tel>
    <Salary>37500</Salary>
  </Employee>

  <Employee EmpNum="123" JobTitle="Director">
    <Name>
      <FirstName>Chris</FirstName>
      <LastName>Williams</LastName>
    </Name>
    <Tel>222-7777-123</Tel>
    <Salary>79500</Salary>
  </Employee>

</Company>
```

```xml
<Employee EmpNum="923" JobTitle="Programmer">
  <Name>
    <FirstName>Joseff</FirstName>
    <LastName>Smith</LastName>
  </Name>
  <Tel>222-7777-923</Tel>
  <Salary>142000</Salary>
</Employee>

<Employee EmpNum="789" JobTitle="Programmer">
  <Name>
    <FirstName>Emily</FirstName>
    <LastName>Smith</LastName>
  </Name>
  <Tel>222-7777-789</Tel>
  <Salary>57000</Salary>
</Employee>

<Employee EmpNum="101" JobTitle="Programmer">
  <Name>
    <FirstName>Thomas</FirstName>
    <LastName>Williams</LastName>
  </Name>
  <Tel>222-7777-101</Tel>
  <DailyRate>425</DailyRate>
</Employee>
```

Parsing XML Data

- You can load XML data from a file
  - Via `ET.parse(xmlFilename)`
  - Returns an `ElementTree`, from which you can get the root element as an `Element` object

```
tree = ET.parse("Company.xml")
root = tree.getroot()
```

- Alternatively you can load XML data from a string
  - Via `ET.fromstring(xmlString)`
  - Returns the root element directly, as an `Element` object

```
root = ET.fromstring("<Company> … </Company>")
```

Getting Element Information

- You can use the following attributes on an Element, to get information about the element in the document:
  - `tag`
  - `attrib`
  - `text`

- Example:

```
print("Root tag name: %s" % root.tag)
print("Root tag attributes: %s" % root.attrib)
print("Root tag text: %s" % root.text)
```

Iterating Over Child Elements

- You can iterate over the children of an `Element`
  - Returns all the child elements as `Element` objects

- Example:

```
for child in root:
    print("  %s, %s, %s" % (child.tag, child.attrib, child.text))
```

Indexing Into Child Elements

- You can index into an Element's children using []
  - Returns a new Element, representing the child you selected
  - You can apply subsequent [] to find one of its children, and so on

- Example:

```
emp1Sal = root[1][2]
print("For employee[1], %s is %s" % (emp1Sal.tag, emp1Sal.text))
```

Finding Specific Child Elements

- You can find child elements that have a specific tag name
  - `find(`*`tagName`*`)` finds the first matching child
  - `findall(`*`tagName`*`)` finds all matching children

- Example:

```
firstEmp = root.find("Employee")

print("\nTelephone numbers for first employee:")

for tel in firstEmp.findall("Tel"):
    print("  %s" % tel.text)
```

Iterating over Descendant Elements

- You can iterate over the descendant elements that have a specific tag name
  - `iter(`*`tagName`*`)`

- Example:

```
print("\nSalaries for all employees:")

for sal in root.iter('Salary'):
    print("  %s" % sal.text)
```

3. Locating Content using XPath

- What is XPath?
- Nodes in an XPath tree
- Supported XPath syntax in Python
- Executing XPath expressions in Python
- Locating attributes
- Locating descendant elements
- XPath and namespaces

What is XPath?

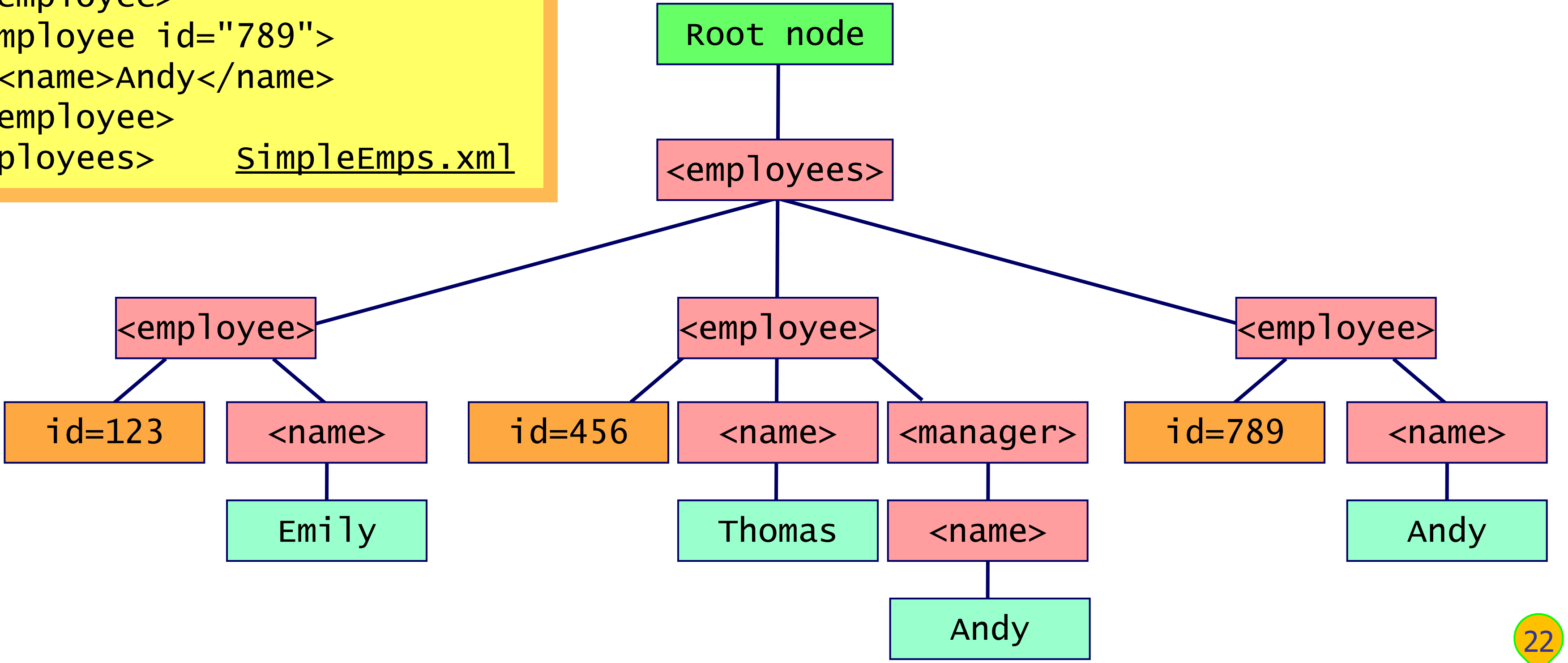
- XPath is a W3C-standard way to locate content in an XML document
  - Maps an XML document into a tree of nodes
  - 7 types of node in an XML document (see next slide)

- Note:
  - All the demo code for this section is in `xpath.py`

Nodes in an XPath Tree

```
<employees>
  <employee id="123">
    <name>Emily</name>
  </employee>
  <employee id="456">
    <name>Thomas</name>
    <manager>
      <name>Andy</name>
    </manager>
  </employee>
  <employee id="789">
    <name>Andy</name>
  </employee>
</employees>    SimpleEmps.xml
```

- 🟩 Root node
- 🟪 Element nodes
- 🟧 Attribute nodes
- 🟩 Text nodes
- ■ Plus 3 other node types…



Supported XPath Syntax in Python

- Python supports a subset of the W3C XPath syntax

Syntax	What does it select...
`tagName`	All child elements with the specified name
`*`	All child elements
`.`	Current node
`//`	Descendant elements, whatever depth
`..`	Parent node
`[@attrib]`	All elements that have the specified attribute
`[@attrib='value']`	All elements that have the specified attribute name/value
`[tagName]`	All elements that have a child element with the specified name
`[tagName='text']`	All elements that have a child element with the specified name/value
`[position]`	All elements at the specified location (starting at 1)

Executing XPath Expressions in Python

- To execute XPath expressions in Python:
  - `find(xpathExpr)`
  - `findAll(xpathExpr)`

- Examples:

```
sals = root.findall("Employee/Salary")

print("All salaries:")
for sal in sals:
    print("  %s" % sal.text)
```

```
salEmp2 = root.find("Employee[2]/Salary")
print("\nSalary for employee[2] is %s" % salEmp2.text)
```

```
names = root.findall("Employee[Salary]/Name")

print("\nFull names of all salaried employees:")
for name in names:
    print("  %s %s" % (name.find("FirstName").text, name.find("LastName").text))
```

Locating Attributes

- You can use attributes in your XPath expressions, of course

- Example:

```
sals = root.findall("Employee[@JobTitle='Programmer']/Salary")

print("\nSalary of all programmers:")
for sal in sals:
    print("  %s" % sal.text)
```

Locating Descendant Elements

- You can use the // operator in your XPath expressions if you need to find descendants

- Example:

```
tels = root.findall(".//Tel")

print("\nTelephone numbers for all employees:")

for tel in tels:
    print("  %s" % tel.text)
```

XPath with Namespaces

- If you want to use XPath to locate XML content that has a namespace, use this syntax to access an element:
    - `'{namespaceUri}localName'`

- Example:
    - See the `XpathWithNamespaces` demo folder

4. Updating XML Data in Python

- Overview
- Typical tasks
- Example

Overview

- The ElementTree API defines numerous methods that allow you to modify the content of the XML document
  - Create a new element
  - Add/modify/remove attributes
  - Modify text content
  - Append/insert/remove child nodes
  - Etc.

- We'll explore some of these capabilities in this section
  - See the demo code in `modifyXml.py`

Typical Tasks

- Create a new element
  - `ET.Element(`*`tagName`*`)`
- Set an attribute on an element
  - *`anElement`*`.set(`*`attrName`*`, `*`attrValue`*`)`
- Set the text content for an element
  - *`anElement`*`.text = `*`textValue`*
- `Append an element to an existing element:`
  - *`existingElement`*`.append(`*`newElement`*`)`

Example

- This example creates everything for a new employee

```
emp = ET.Element("Employee")
emp.set("EmpNum", "ZZ123")
emp.set("JobTitle", "Rollercoaster designer")
root.append(emp)

firstname = ET.Element("FirstName")
firstname.text = "Zak"
lastname = ET.Element("LastName")
lastname.text = "Thunderbolt"

name = ET.Element("Name")
name.append(firstname)
name.append(lastname)
emp.append(name)

tel = ET.Element("Tel")
tel.text = "222-7777-999"
emp.append(tel)

sal = ET.Element("Salary")
sal.text = "250000"
emp.append(sal)

tree.write("UpdatedCompany.xml")
```

Any Questions?