XML Processing in Python

Objectives

At the end of this second session, you will be able to understand:

- XML Processing in Python

XML Processing in Python

Python XML processing

- XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.
- XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of the operating system and/or developmental language.

XML Parser Architectures and APIs

The Python standard library provides a minimal but useful set of interfaces to work with XML.
The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

- **Simple API for XML (SAX):** Here, you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.
- **Document Object Model (DOM) API:** This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

Python XML processing (contd.)

- SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.
- SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.
- For all our XML code examples, let's use a simple XML file movies.xml as an input:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
    <type>War, Thriller</type>
    <format>DVD</format>
    <year>2003</year>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Talk about a US-Japan war</description>
```

Python XML Processing (contd.)

```
</movie>
<movie title="Transformers">
    <type>Anime, Science Fiction</type>
    <format>DVD</format>
    <year>1989</year>
    <rating>R</rating>
    <stars>8</stars>
    <description>A schientific fiction</description>
</movie>
<movie title="Ishtar">
    <type>Comedy</type>
    <format>VHS</format>
    <rating>PG</rating>
    <stars>2</stars>
    <description>Viewable boredom</description>
</movie>
</collection>
```

Parsing XML with SAX APIs

- SAX is a standard interface for event-driven XML parsing. Parsing XML with SAX generally requires you to create your own ContentHandler by subclassing xml.sax.ContentHandler.
- Your ContentHandler handles the particular tags and attributes of your flavor(s) of XML. A ContentHandler object provides methods to handle various parsing events. Its owning parser calls ContentHandler methods as it parses the XML file.
- The methods startDocument and endDocument are called at the start and the end of the XML file. The method characters(text) is passed character data of the XML file via the parameter text.
- The ContentHandler is called at the start and end of each element. If the parser is not in namespace mode, the methods **startElement(tag, attributes)** and **endElement(tag)** are called; otherwise, the corresponding methods **startElementNS** and **endElementNS** are called. Here, tag is the element tag, and attributes is an Attributes object.

Parsing XML with SAX APIs (contd.)

Here are other important methods to understand before proceeding.

The make parser Method:

The following method creates a new parser object and returns it. The parser object created will be of the first parser type the system finds.

```
xml.sax.make_parser( [parser_list] )
```

Here are the details of the parameters:

parser_list: The optional argument consisting of a list of parsers to use which must all implement the make_parser method.

Parsing XML with SAX APIs (contd.)

The parse Method:

Following method creates a SAX parser and uses it to parse a document.

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler])
```

Here are the details of the parameters:

xmlfile: This is the name of the XML file to read from.

contenthandler: This must be a ContentHandler object.

errorhandler: If specified, errorhandler must be a SAX ErrorHandler object.

Parsing XML with SAX APIs (contd.)

Example:

```
#!/usr/bin/python

import xml.sax
#event based Simple API for SML parser

class MovieHandler( xml.sax.ContentHandler ):
#subclass of xml.sax.ContentHandler super class

        count=0;

        def __init__(self):     #constructor

              self.CurrentData = ""

              self.type = "“

              self.format = ""

              self.year = ""

              self.rating = ""

              self.stars = ""

              self.description = ""
```

Parsing XML with SAX APIs (contd.)

```
def startElement(self, tag, attributes):
# Call when an element starts

    self.CurrentData = tag

    if tag == "movie":

        print "*****Movie*****"

        title = attributes["title"]

        print "Title:", title
```

```
def endElement(self, tag):
# # Call when an elements ends

    if self.CurrentData == "type":

        print "Type:", self.type

    elif self.CurrentData == "format":

        print "Format:", self.format

    elif self.CurrentData == "year":

        print "Year:", self.year

    elif self.CurrentData == "rating":

        print "Rating:", self.rating
```

Parsing XML with SAX APIs (contd.)

```
elif self.CurrentData == "stars":
          print "Stars:", self.stars
      elif self.CurrentData == "description":
          print "Description:", self.description
      self.CurrentData = "“
```

```
# Call when a character is read
def characters(self, content):
      if self.CurrentData == "type":
          self.type = content
      elif self.CurrentData == "format":
          self.format = content
      elif self.CurrentData == "year":
          self.year = content
      elif self.CurrentData == "rating":
          self.rating = content
```

Parsing XML with SAX APIs (contd.)

```
elif self.CurrentData == "stars":
    self.stars = content
    if(int(self.stars))>8:
        MovieHandler.count+=1;
elif self.CurrentData == "description":
    self.description = content
```

```
if ( __name__ == "__main__"):
    # create an XMLReader
    parser = xml.sax.make_parser()
    # turn off namepsaces
    parser.setFeature(xml.sax.handler.feature_namespaces, 0)
```

Persistent

Parsing XML with SAX APIs (contd.)

```
# override the default ContextHandler

Handler = MovieHandler()    #costructor

parser.setContentHandler( Handler )


parser.parse("movies.xml")#parsing of movies.xml document is taking place


print "Count = ", MovieHandler.count
```

Parsing XML with SAX APIs (contd.)

Output

```
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
```

```
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Year: 1989
Rating: R
Stars: 8
Description: A schientific fiction
```

Parsing XML with DOM APIs

- The Document Object Model ("DOM") is a cross-language API from the World Wide Web Consortium (W3C) for accessing and modifying XML documents.
- The DOM is extremely useful for random-access applications. SAX only allows you a view of one bit of the document at a time. If you are looking at one SAX element, you have no access to another.
- Here is the easiest way to quickly load an XML document and to create a minidom object using the xml.dom module. The minidom object provides a simple parser method that quickly creates a DOM tree from the XML file.
- The sample phrase calls the parse(file [,parser]) function of the minidom object to parse the XML file designated by file into a DOM tree object.

Parsing XML with DOM APIs (contd.)

```
#!/usr/bin/python

#DOM =Document object model

from xml.dom.minidom import parse

import xml.dom.minidom    #random access parsing

# Open XML document using minidom parser

DOMTree = xml.dom.minidom.parse("movies.xml")

collection = DOMTree.documentElement   #returns root element

if collection.hasAttribute("shelf"):

   print "Root element : %s" % collection.getAttribute("shelf")
```

Parsing XML with DOM APIs (contd.)

```
# Get all the movies in the collection

movies = collection.getElementsByTagName("movie")
                #list

for movie in movies:   # Print detail of each movie.

        print "*****Movie*****"

        if movie.hasAttribute("title"):

              print "Title: %s" % movie.getAttribute("title")

type = movie.getElementsByTagName('type')[0]

print "Type: %s" % type.childNodes[0].data

format = movie.getElementsByTagName('format')[0]

print "Format: %s" % format.childNodes[0].data

rating = movie.getElementsByTagName('rating')[0]

print "Rating: %s" % rating.childNodes[0].data

description = movie.getElementsByTagName
                      ('description')[0]

print "Description: %s" %
               description.childNodes[0].data
```

Parsing XML with DOM APIs (contd.)

Output

```
Root element: New Arrivals

*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Rating: PG
Description: Talk about a US-Japan war

*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A schientific fiction
```

Assignments

1. Read given XML file movies.xml. Print the total count of movie details stored in it. Also display all movie details.

Summary

With this we have come to the end of our session, where we discussed about:

- XML Processing in Python

In the next session we will discuss about

- Writing Web Apps with Python

Reference material

- http://www.tutorialspoint.com/python
- http://www.learnpython.org/
- http://docs.python.org/2/tutorial/
- https://docs.python.org/2/tutorial/stdlib.html
- https://docs.python.org/2/tutorial/stdlib2.html

Persistent

Questions

Key contacts

Sakshi Jamgaonkar

sakshi_jamgaonkar@persistent.com

Asif Immanad

asif_immanad@persistent.co.in

Persistent

Persistent

Thank you!