INDEX

ABSTRACT

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

Simple API for XML (SAX) is a lexical, event-driven API in which a document is read serially and its contents are reported as callbacks to various methods on a handler object of the user's design. SAX is fast and efficient to implement, but difficult to use for extracting information at random from the XML, since it tends to burden the application author with keeping track of what part of the document is being processed. It is better suited to situations in which certain types of information are always handled the same way, no matter where they occur in the document.

A parser is a program, usually part of a compiler, that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or options) that can then be managed by other programming (for example, other components in a compiler). A parser may also check to see that all input has been provided that is necessary.

An XML Parser is a program which breaks the structure of the XML Document and forms modules of the Code while going through the whole XML Document Syntactically.

1. INTRODUCTION

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers. The goal of a parser is to transform XML into a readable code.

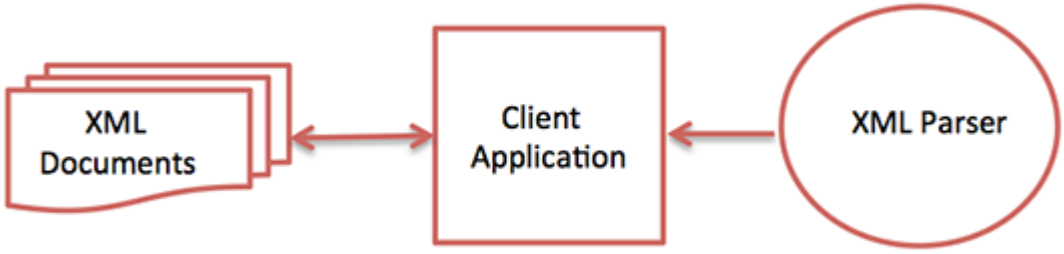
2. BACKGROUND OF WORK

We gone through the syntax and the structure of XML. We collected some information about parsers and how to implement their functionality in scanning XML document. We gone through some sample codes on how to extract information from a normal document. We have implemented the idea above for XML documents. Basically wherever the parser finds a match of the string(s) provided, it will collect all the data related to the string and store it in a separate text file for each string provided.

3. OVERVIEW OF THE WORK

3.1 **Problem Description**: We are required to extract information about specialists in given field(or any sample XML document) by using XML document parser.

3.2 **Diagram/Working Model**: Following diagram shows how XML parser interacts with XML document:



3.3 Design Description:

- Information about the specialists in a given field will reside in an XML document.
- The document will be scanned line by line.
- Then compare the scanned information from the input document with strings (e.g: orthodontist, software engineer etc.).
- If a match is found then that value and the information related to it will be written on a separate text file.
- Separate text files containing the list of people in same field will be created**.**

4. SAX PARSERS

SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API developed by the XML-DEV mailing list. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

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.

5. IMPLEMENTATION

5.1. Description of Modules / Programs

```
pars = open(filename)                          #Opening XML file from INPUT
class Handler( xml.sax.ContentHandler ):  #Handles the particular tags and attributes
                                                                    of the XML Doc.
 def startElement(self, tag, attributes):      #Call when an element or tag starts
 def endElement(self, tag):                    #Call when an elements ends
 def characters(self, content):                #Call when a character is read
 os.startfile('parsed.txt')                      #Open the file where parsed contents are
                                                                    stored : PARSED.txt
```

5.2. Source Code

```
import xml.sax                                  #SAX (Simple API for XML) is an event-
driven algorithm for parsing XML documents
import os                                       #For opening parsed.txt
import re
import sys
from time import sleep                          #Typewritter text
def tag(str,a):
   for i in range(a,len(str)):
      if  str[i]=='>' or str[i]=='"' or str[i]==' ' or str[i]=='=':
         break
   return i;
attr=[]
l=0
filename = input('Enter a file name: ')
pars = open(filename)   #Opening XML file from INPUT
parsed = open("attributes.txt","w+")    #Parsed Content of the XML Doc to be stored here
for str in pars.readlines():
   str=str.lstrip()
   if str[0]=='<':
```

```
        i=tag(str,1)
        if str[i]==' ':
            w=i
            j=tag(str,w+1)
            parsed.write(str[1:i].rstrip()+"\n"+str[w+1:j].rstrip()+"\n")
            #attr[1]=str[1:i]
        else:
            parsed.write(str[1:i]+"\n")
    else:
        print("\n\nSyntax Error !!!!")
        input()
        exit(0)
pars.close()
parsed.close()
pars = open("parsed.txt","w+")                    #Parsed Content of the XML Doc to be
stored here
parsed = open("attributes.txt","r")
parsed.readline().strip()
lis = [parsed.readline().strip()]
w='/'+lis[0]+'\n'
for q in parsed.readlines():
   if q==w:
      break
   else:
      lis.append(q[0:-1])
if len(lis)<8:
   for i in range(0,(8-len(lis))):
      lis.append("")
parsed.close()
parsed = open("attributes.txt","w")
parsed.seek(0)
parsed.truncate()
parsed.writelines(["%s\n" % item  for item in lis])
parsed.close()
```

```
class Handler( xml.sax.ContentHandler ):                    #Handles the particular tags and attributes of the XML Doc.
  def __init__(self):
    self.CurrentData = ""
    self.type = ""
    self.format = ""
    self.year = ""
    self.rating = ""
    self.stars = ""
    self.description = ""

  def startElement(self, tag, attributes):                  #Call when an element or tag starts
    self.CurrentData = tag                                  #Reads evey word of XML Doc
    if tag == lis[0]:
      print("\n\n")
      title = attributes[lis[1]]
      print (lis[1],"\t\t  :  ", title)
      pars.write("\n\n"+lis[1]+" : "+title+"\n")

  def endElement(self, tag):                                #Call when an elements ends
    if self.CurrentData == lis[2]:
      print (lis[2],"\t\t  :  ", self.type)
      pars.write(lis[2]+" : "+self.type+"\n")
    elif self.CurrentData == lis[3]:
      print (lis[3],"\t\t  :  ", self.format)
      pars.write(lis[3]+" : "+self.format+"\n")
    elif self.CurrentData == lis[4]:
      print (lis[4],"\t\t  :  ", self.year)
      pars.write(lis[4]+" : "+self.year+"\n")
    elif self.CurrentData == lis[5]:
      print (lis[5],"\t\t  :  ", self.rating)
      pars.write(lis[5]+" : "+self.rating+"\n")
    elif self.CurrentData == lis[6]:
      print (lis[6],"\t\t  :  ", self.stars)
```

```
        pars.write(lis[6]+" : "+self.stars+"\n")
      elif self.CurrentData == lis[7]:
        print (lis[7],"  :  ", self.description + "\n")
        pars.write(lis[7]+" : "+self.description+"\n\n")
      self.CurrentData = " "

   def characters(self, content):                    #Call when a character is read
      if self.CurrentData == lis[2]:
        self.type = content
      elif self.CurrentData == lis[3]:
        self.format = content
      elif self.CurrentData == lis[4]:
        self.year = content
      elif self.CurrentData == lis[5]:
        self.rating = content
      elif self.CurrentData == lis[6]:
        self.stars = content
      elif self.CurrentData == lis[7]:
        self.description = content



if ( __name__ == "__main__"):
   parser = xml.sax.make_parser()
   parser.setFeature(xml.sax.handler.feature_namespaces, 0)
   Handler = Handler()                          #Initializes the above defined ContextHandler
   parser.setContentHandler( Handler )
   if filename[-4:]!=".xml":
      print("Not a valid XML Doc")
      exit();
   parser.parse(filename)                        #Opening XML file from INPUT to PARSE
pars.close()
input()
print("_____”)
q=("\nPress Enter to open the Parsed File : " + pars.name + "\n")
```

```
for x in q:
  print(x,end='')
  sleep(0.01)
input()
for x in ("\nOPENING FILE : " + pars.name + "\n"):
  print(x,end='')
  sleep(0.009)
os.startfile('parsed.txt')                    #Open the file where parsed contents are
stored : PARSED.txt
```

5.3. Test Cases

E1.xml

```
<collection>
<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>
</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="Trigun">
  <type>Anime, Action</type>
  <format>DVD</format>
```

```
  <episodes>4</episodes>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
  <type>Comedy</type>
  <format>VHS</format>
  <rating>PG</rating>
  <stars>2</stars>
  <description>Viewable boredom</description>
</movie>
</collection>
```

E2.xml

```
<sites>
                                        <site rating="5">
                                                <title>Dave Eddy</title>
                                                <url>http://www.daveeddy.com</url>
                                        </site>
                                        <site rating="4">
                                                <title>Lights and Shapes</title>
                                                <url>http://lightsandshapes.com</url>
                                        </site>
                                        <site rating="3" nsfw="false">
                                                <title>Duck Duck Go</title>
                                                <url>http://www.duckduckgo.com</url>
                                        </site>
</sites>
```

E3.xml

```
<catalog>
```

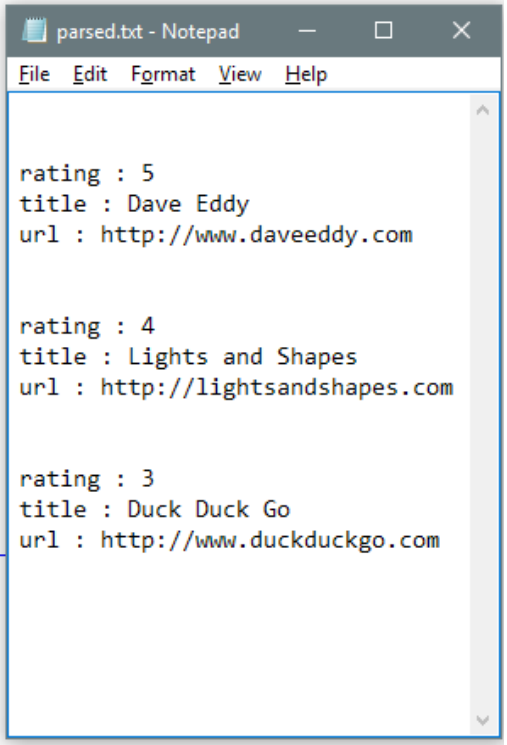
```
<book id="bk101">
   <author>Gambardella, Matthew</author>
   <title>XML Developer's Guide</title>
   <genre>Computer</genre>
   <price>44.95</price>
   <publish_date>2000-10-01</publish_date>
   </book>
<book id="bk102">
   <author>Ralls, Kim</author>
   <title>Midnight Rain</title>
   <genre>Fantasy</genre>
   <price>5.95</price>
   <publish_date>2000-12-16</publish_date>
</book>
<book id="bk103">
   <author>Corets, Eva</author>
   <title>Maeve Ascendant</title>
   <genre>Fantasy</genre>
   <price>5.95</price>
   <publish_date>2000-11-17</publish_date>
</book>
<book id="bk104">
   <author>Corets, Eva</author>
   <title>Oberon's Legacy</title>
   <genre>Fantasy</genre>
   <price>5.95</price>
   <publish_date>2001-03-10</publish_date>
</book>
<book id="bk105">
   <author>Corets, Eva</author>
   <title>The Sundered Grail</title>
   <genre>Fantasy</genre>
   <price>5.95</price>
   <publish_date>2001-09-10</publish_date>
```

```
</book>
<book id="bk106">
  <author>Randall, Cynthia</author>
  <title>Lover Birds</title>
  <genre>Romance</genre>
  <price>4.95</price>
  <publish_date>2000-09-02</publish_date>
</book>
<book id="bk107">
  <author>Thurman, Paula</author>
  <title>Splish Splash</title>
  <genre>Romance</genre>
  <price>4.95</price>
  <publish_date>2000-11-02</publish_date>
</book>
<book id="bk108">
  <author>Knorr, Stefan</author>
  <title>Creepy Crawlies</title>
  <genre>Horror</genre>
  <price>4.95</price>
  <publish_date>2000-12-06</publish_date>
</book>
<book id="bk109">
  <author>Kress, Peter</author>
  <title>Paradox Lost</title>
  <genre>Science Fiction</genre>
  <price>6.95</price>
  <publish_date>2000-11-02</publish_date>
</book>
<book id="bk110">
  <author>O'Brien, Tim</author>
  <title>Microsoft .NET: The Programming Bible</title>
  <genre>Computer</genre>
  <price>36.95</price>
```

```
    <publish_date>2000-12-09</publish_date>
  </book>
  <book id="bk111">
    <author>O'Brien, Tim</author>
    <title>MSXML3: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-01</publish_date>
  </book>
  <book id="bk112">
    <author>Galos, Mike</author>
    <title>Visual Studio 7: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>49.95</price>
    <publish_date>2001-04-16</publish_date>
  </book>
</catalog>
```

5.4. Execution Snapshots

```
>>>
Enter a file name: e2.xml


rating            :   5
title             :   Dave Eddy
url               :   http://www.daveeddy.com


rating            :   4
title             :   Lights and Shapes
url               :   http://lightsandshapes.com


rating            :   3
title             :   Duck Duck Go
url               :   http://www.duckduckgo.com

------------------------------------------------------------

Press Enter to open the Parsed File : parsed.txt


OPENING FILE : parsed.txt
>>>
```



```
Enter a file name: e1.xml


title              :   Enemy Behind
type               :   War, Thriller
format             :   DVD
year               :   2003
rating             :   PG
stars              :   10
description    :   Talk about a US-Japan war



title              :   Transformers
type               :   Anime, Science Fiction
format             :   DVD
year               :   1989
rating             :   R
stars              :   8
description    :   A schientific fiction



title              :   Trigun
type               :   Anime, Action
format             :   DVD
rating             :   PG
stars              :   10
description    :   Vash the Stampede!



title              :   Ishtar
type               :   Comedy
format             :   VHS
rating             :   PG
stars              :   2
description    :   Viewable boredom
```

6. CONCLUSION

This code provides a simple and effective interface to parse XML documents using Python Language and SAX Parsing Techniques.

The developed XML parser shows that the functional approach accomplishes the task of parsing and validating XML by using fewer lines of code and producing a very short and compact program in contrast to tiresome languages.

7. REFERENCES

1. Tutorialspoint www.tutorialspoint.com
2. Python Docs https://docs.python.org/3/library/xml.sax.html
3. W3shools http://www.w3schools.com/xml/xml_parser.asp