

**Project Report**

**FOP II**

|  |  |
| --- | --- |
| **NAME** | **CMS** |
| Burhan Baig | 460945 |
| Haseeb Aamir | 468333 |
| Minahil Shahzad | 466695 |
| Syed Ali Naqi | 455426 |
| 1. Ahmed Raza | 467253 |

## Introduction

This project focuses on developing an RSS feed filter that processes news stories from various sources, applies user-defined triggers to filter relevant items, and displays the filtered news in a GUI. The system is modular, with distinct classes and functions dedicated to specific tasks such as fetching and parsing RSS feeds, defining triggers based on various criteria, and filtering and displaying news stories.

## Logic Behind the Code

## Code and its Explanation

This code is organized into separate blocks, each serving a specific function and responsibility. This structure ensures smooth program execution and facilitates easy debugging.

**Block 1: Class Definitions and Basic Structures**

import re

import feedparser

import string

import time

import threading

from datetime import datetime

import pytz

import tkinter as tk

from tkinter import Frame, Scrollbar, StringVar, Label, Text, Button

from bs4 import BeautifulSoup

class NewsStory:

def \_\_init\_\_(self, g, t, d, l, p):

self.G = g

self.T = t

self.D = d

self.L = l

self.P = p

def GetG(self):

return self.G

def GetT(self):

return self.T

def GetD(self):

return self.D

def GetL(self):

return self.L

def GetP(self):

return self.P

**Explanation:**

Imports necessary libraries and modules and defines the `NewsStory` class to store information about news stories, such as GUID, title, description, link, and published date and provides methods to retrieve each piece of information from the `NewsStory` object.

**Block 2: Trigger Classes**

class Trigger:

def Eval(self, s):

raise NotImplementedError

class PhraseTrigger(Trigger):

def \_\_init\_\_(self, phrase):

self.Phrase = phrase.lower()

def IsIn(self, text):

text = text.lower()

for p in string.punctuation:

text = text.replace(p, ' ')

words = text.split()

phrase\_words = self.Phrase.split()

return ' '.join(words).find(' '.join(phrase\_words)) != -1

class TitleTrigger(PhraseTrigger):

def Eval(self, s):

return self.IsIn(s.T)

class DescriptionTrigger(PhraseTrigger):

def Eval(self, s):

return self.IsIn(s.D)

class TimeTrigger(Trigger):

def \_\_init\_\_(self, time):

self.T = datetime.strptime(time, "%d %b %Y %H:%M:%S").replace(tzinfo=pytz.timezone("EST"))

class BeforeTrigger(TimeTrigger):

def Eval(self, s):

return s.GetP().replace(tzinfo=pytz.timezone("EST")) < self.T

class AfterTrigger(TimeTrigger):

def Eval(self, s):

return s.GetP().replace(tzinfo=pytz.timezone("EST")) > self.T

class NotTrigger(Trigger):

def \_\_init\_\_(self, t):

self.T = t

def Eval(self, s):

return not self.T.Eval(s)

class AndTrigger(Trigger):

def \_\_init\_\_(self, t1, t2):

self.T1 = t1

self.T2 = t2

def Eval(self, s):

return self.T1.Eval(s) and self.T2.Eval(s)

class OrTrigger(Trigger):

def \_\_init\_\_(self, t1, t2):

self.T1 = t1

self.T2 = t2

def Eval(self, s):

return self.T1.Eval(s) or self.T2.Eval(s)

**Explanation:**

Defines various trigger classes that will be used to filter news stories based on criteria like phrases in titles or descriptions, specific times, or logical combinations (AND, OR, NOT).

**Block 3: Filtering and Configuration Reading**

```python

def Filter(s, tlist):

filtered\_stories = []

for story in s:

if any(trigger.Eval(story) for trigger in tlist):

filtered\_stories.append(story)

return filtered\_stories

def ReadConfig(filename):

trigger\_file = open(filename, 'r')

lines = []

for line in trigger\_file:

line = line.rstrip()

if not (len(line) == 0 or line.startswith('//')):

lines.append(line)

dicty = {

'TITLE': TitleTrigger,

'DESCRIPTION': DescriptionTrigger,

'AND': AndTrigger,

'OR': OrTrigger,

'NOT': NotTrigger,

'AFTER': AfterTrigger,

'BEFORE': BeforeTrigger

}

trigger\_dict = {}

trigger\_list = []

for line in lines:

data = re.split(",", line)

if data[0] != 'ADD':

if data[1] in ['TITLE', 'DESCRIPTION', 'AFTER', 'BEFORE', 'NOT']:

trigger\_dict[data[0]] = dicty[data[1]](data[2])

elif data[1] in ['AND', 'OR']:

trigger\_dict[data[0]] = dicty[data[1]](trigger\_dict[data[2]], trigger\_dict[data[3]])

elif data[0] == 'ADD':

trigger\_list.extend(trigger\_dict[t] for t in data[1:])

return trigger\_list

**Explanation:**

`Filter` function takes a list of stories and a list of triggers, returning stories that match any trigger. `ReadConfig` function reads a configuration file to create and return a list of triggers based on the file's instructions.

**Block 4: Fetching and Processing News Stories**

def Process(url):

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

g = entry.guid

t = entry.title

l = entry.link

d = entry.description

p = entry.published

try:

p = datetime.strptime(p, "%a, %d %b %Y %H:%M:%S %Z")

p = p.replace(tzinfo=pytz.timezone("GMT"))

except ValueError:

p = datetime.strptime(p, "%a, %d %b %Y %H:%M:%S %z")

newsStory = NewsStory(g, t, d, l, p)

ret.append(newsStory)

return ret

**Explanation:**

`Process` function fetches news stories from a given RSS feed URL, parses them into `NewsStory` objects, and returns a list of these objects.

**Block 5: GUI Update and Main Thread**

SLEEPTIME = 120

def UpdateGui(master, cont, scrollbar):

try:

triggerlist = ReadConfig('triggers.txt')

guidShown = []

while True:

print("Polling . . .", end=' ')

stories = Process("http://news.google.com/news?output=rss")

print(f"Number of fetched stories: {len(stories)}\nFiltered Stories:")

filtered\_stories = Filter(stories, triggerlist)

print(f"Number of filtered stories: {len(filtered\_stories)}")

cont.delete('1.0', tk.END)

for story in filtered\_stories:

cont.insert(tk.END, f"Title: {story.GetT()}\n")

cont.insert(tk.END, f"Description: ")

# Parse HTML description

soup = BeautifulSoup(story.GetD(), 'html.parser')

description\_text = soup.get\_text(separator='\n')

cont.insert(tk.END, description\_text + "\n")

cont.insert(tk.END, f"Link: {story.GetL()}\n\n")

cont.insert(tk.END, "---------------------------------------------------------------\n\n")

scrollbar.config(command=cont.yview)

print("Sleeping...")

time.sleep(SLEEPTIME)

except Exception as e:

print("An error occurred:", e)

def MainThread(master):

frame = Frame(master)

frame.pack(side=tk.BOTTOM)

scrollbar = Scrollbar(master)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

t = "Google & Yahoo Top News"

title = StringVar()

title.set(t)

ttl = Label(master, textvariable=title, font=("Times New Roman", 18))

ttl.pack(side=tk.TOP)

cont = Text(master, font=("Times New Roman", 14), yscrollcommand=scrollbar.set, wrap=tk.WORD)

cont.pack(side=tk.BOTTOM)

button = Button(frame, text="Exit", command=root.destroy)

button.pack(side=tk.BOTTOM)

gui\_thread = threading.Thread(target=UpdateGui, args=(master, cont, scrollbar))

gui\_thread.daemon = True

gui\_thread.start()

if \_\_name\_\_ == '\_\_main\_\_':

root = tk.Tk()

root.title("Some RSS parser")

MainThread(root)

root.mainloop()

**Explanation:**

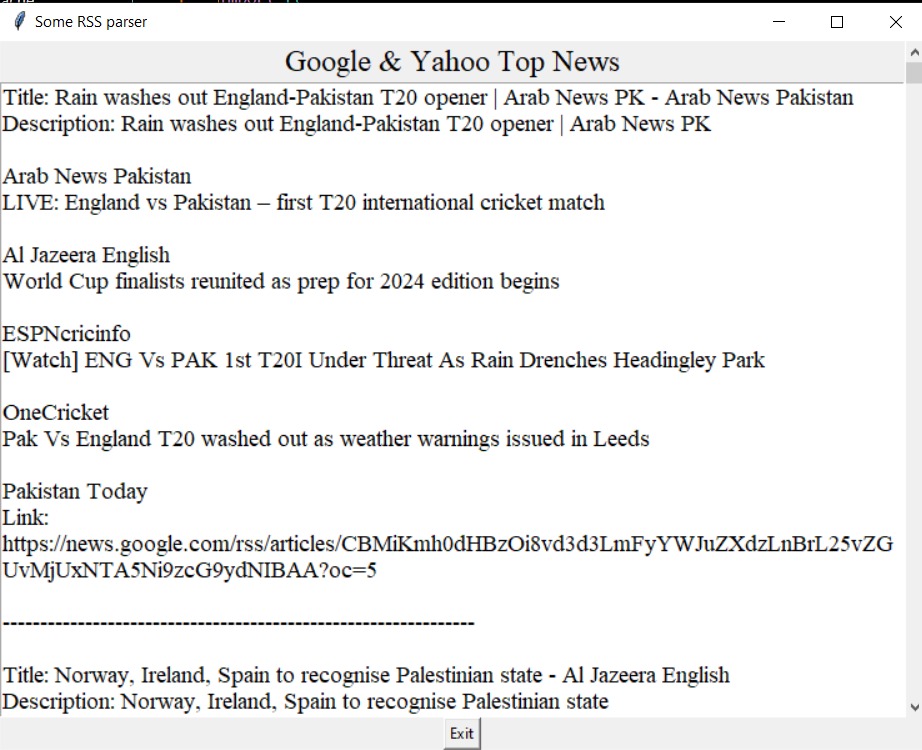
`UpdateGui` function updates the GUI with news stories, polling every `SLEEPTIME` seconds. `MainThread` function sets up the Tkinter GUI with a title, text area, scrollbar, and an exit button. Main part of the script starts the Tkinter main loop and runs the GUI.

The code utilizes object-oriented programming (OOP) principles with classes like `NewsStory` representing news stories and trigger classes inheriting from `Trigger` for evaluating news based on criteria.

Parsing RSS feeds is done using the `feedparser` library, while `BeautifulSoup` parses HTML content in news descriptions. Text normalization, date/time parsing with `datetime` and `pytz`, and file I/O operations with regular expressions are included. The GUI is built with Tkinter, using multithreading for responsiveness.

Error handling is ensured with `try`/`except` blocks, and the code is modular and reusable, with distinct functions for filtering, reading configurations, and processing data. These techniques result in an efficient and maintainable RSS parsing and filtering application.

## Output



## Conclusion

The collaborative effort on the RSS feed filter project employs a modular design to effectively process and filter news stories using user-defined triggers. By distributing responsibilities among team members, we have ensured the smoothness and quality implementation of each block of code, including fetching and parsing RSS feeds, defining and evaluating triggers, and filtering and displaying news stories. This project showcases our capability to develop a functional and user-friendly application that meets defined requirements.It also shows that Object oriented programming is a necessary tool used in making any type of code. The resulting system serves as a potent tool for users, allowing them to stay updated on relevant news stories filtered through customizable triggers.