5/19/2021

A Tool for IWXXM XML Format Data

Supervisor: Gaetan DURY, OF-2

Intern/Documentation: Naime Celik

SHAPE SEM J2 GEOMETOC

Table of Content

Table of Content 1
Table of Figures 1
Introduction 2
External Code Sources 2
The Restrictions and Problems Regarding the Test Environment 2
Calculation of ColorState 3
CrossWindAlert Calculation 4
Running Application with Python VirtualEnv 6
Running Swagger Web Application with Docker 12
References 20
Appendix 21

Table of Figures

Figure 1 Sample Airport Station JSON data retrieved from AVWX Rest API for LIZC 5
Figure 2 Cross wind representation with Runway 280 °, Wind direction 60 °, Wind Speed will result 3.8567 as crosswind component. (Figure retrieved from https://aerotoolbox.com/crosswind/) 6
Figure 3 The –h option show available uses 7
Figure 4 Results of running command will show the name of created file for created XML 7
Figure 5 Resulting XML file can be viewed though browser 8
Figure 6 Results of running command will show the name of created file for modified XML 9
Figure 7 Modified XML file after calculation 10
Figure 8 The included part of the xml parameters under remarks 10
Figure 9 Result of METAR string search for a given station designator 11
Figure 10 The docker file to run the application in the working directory 12
Figure 11 The necessary files needs to be in same directory of the Dockerfile 13
Figure 12 The swagger web service created for Metar Data conversion and XML Modification 14
Figure 13 Expanded view of the METAR to XML Conversion service 14
Figure 14 The view of the service method before execution 15
Figure 15 Response body will provide link to download converted XML file 16
Figure 16 Downloaded XML file from response body 17
Figure 17 View from XMLModify service after execution 18
Figure 18 Downloaded XML file from response body 19
Figure 19 Function to retrieve airport station json files containing runway information from AVWX Rest API 21
Figure 20 Function to create Aerodromes tbl and db file 23
Figure 21 Docker file used for creating Docker container for testing cx_freeze for console application. (Unsuccessful due to segment fault error in Docker container for Linux) 24
Figure 22 setup.py file to freeze python as executable (failed in Linux Docker container but works in windows –do not include bin files for windows) 25

Introduction

A python console application and a swagger based web application has been created to provide conversion between METAR string and XML file as well as calculation of ColorState and CrossWindAlert.

The script created for console application can be used;

1. To convert a user given METAR string to IWXXM format XML file.
2. To modify an existing XML document in IWXXM format and, add ColorState and CrossWindAlert with "iwxxm-nato" XML elements under remarks by providing a file given in a URL or a direct path.
3. Additionally, it can search multiple hourly message files to find the recent message of the user given station designator for all METAR string within each bulletin file (SAXXXX), with the optional parameter to generate an IWXXM format XML file. (not applicable with the web service application)

External Code Sources

The script uses some external components and sources that are directly used or modified.

1. avwx-engine: Aviation weather report parsing library that allows to both parsing TAC format data and requests METAR data and station information from AVWX REST API (2021). This is an open-source and publicly available API providing weather data within the daily limit. The avwx-engine repository is maintained by DuPont (2021).
2. GIFTs: The python desktop software created by NOAA to generate IWXXM From TAC (Oberfield,2021). The software is created as a stand-alone desktop application. To conversion in the console application, its "demo.py" file is modified in a way that is accepting any METAR string to generate an IWXXM file.

The GIFTs repository does not have the full aerodrome database and it requires generating a new one for each airport that will be queried. GML attributes contain the coordinates for each airport for the generated XML file. This information is acquired from a locally generated aerodrome database. The created functions to generate .tbl to .db from the airport.csv file were included but the elevation information of each airport was assumed to be 0 as the airport data found online does not contain this information. The airport data can be replaced with a more precise one for production use.

The Restrictions and Problems Regarding the Test Environment

As the use of avwx-engine and many of the python packages and their dependencies requires the use of Python version with 3.7 and above. Due to the unavailability of installing this version of python in NATO site in unclassified and classified desktops, the development of this script was done in the Google CoLab.

The tool was needed to tested in NCIA, however, in the beginning, it was informed that there will be no internet access to install dependencies. The freezing of the tool could be solution for the use of tool with packages.

To create standalone executables from Python scripts, the testing of the console application was also done in the NCIA unclassified laptop (windows). cx_feeze (2021) python module can create an executable file that runs in any operating system and it has been tested. Although the creation of .exe

file for the windows operation system was successful, the necessary executable should be created to run in a Linux environment where the NAMIS METAR data can be accessed. The frozen application under the Windows operating system cannot be run in Linux RedHat 6.5 virtual machine provided in NCIA. It requires another Linux machine to be frozen as Duarte (2021) has stated in the documentation of the cx_freeze.

A docker container was also tested to create the freeze application with docker image ("centos/python-38-centos7) from Docker HUB, which is the closest version to RedHat 6.5. Due to an unknown segment fault error, the frozen version was not created.

It is also seen that even if the virtual machine with RedHat 6.5 is given internet access to retrieve necessary packages through yum, the highest python version yum provides will be Python 3.4. It has been depreciated since 2019 March. The console application created cannot be run in this version of Python. A Docker container with Centos 6 was used to build Python 3.7 from scratch to simulate the environment. Although, Python 3.7 was built successfully in the Docker container, any attempt to retrieve python packages with pip encountered SSLError, which could not be resolved.

As a result, the testing of this console application could not be done in the virtual machine (Linux- RedHat 6.5) provided in NCIA, and the sample data provided in this virtual machine cannot be carried outside due to security reasons.

Due to mentioned issues, a new virtual machine with temporary access to the internet was provided. Python 39 has been built from binary for a new virtual machine running the latest Centos version (commands can be found in the appendix). The testing was done with the METAR folder containing METAR bulletins in the server. We were able to retrieve the latest METAR string for the specified station and were able to generate an IWXXM format file with an additional parameter.

Calculation of ColorState

ColorState class provides a necessary calculation for the color state from either METAR string or IWXXM format XML file. The color states are assigned based on the visibility and state of cloud layers. Once the visibility and cloud layer information was parsed from either METAR string or XML file.

There is/are;

1. visibility info but not cloud layer info then assign color state based on the visibility
2. no visibility info but cloud layer info then assign color state based on the cloud layer
3. both visibility and cloud layer info then assign color state based on both information

Only broken, overcast or scattered of those with the lowest cloud layer was used for evaluation. If the data contain CAVOK then the color state was assigned as BLU.

Table 1 Conditions for Color State

CH = Cloud Height (ft)

V = Visibility (m)

BLU		CH > 2500	-		V > 8000
WHT	2500 >=	CH > 1500	-	8000 >=	V > 5000
GRN	1500 >=	CH > 700	-	5000 >=	V > 3700
YLO1	700 >=	CH > 500	-	3700 >=	V > 2500
YLO2	500 >=	CH > 300	-	2500 >=	V > 1600
AMB	300 >=	CH > 200	-	1600 >=	V > 800
RED	200 >=	CH	-	800 >=	V

ColorState class has two methods, 1) calculateColorState for both METAR raw string and XML format data 2) createModifiedIwxxmFileforColorState for XML format data.

CrossWindAlert Calculation

CrossWind class handles crosswind component calculations, for crosswind component calculation, it requires airport station information that can be provided as a JSON file where many of the station information was retrieved beforehand or it can be retrieved from AVWX Rest API “getMetarInfo” function. The function returns a .json format data where the runway information can be accessed.

```
{
  "city": "Sigonella (CT)",
  "country": "IT",
  "elevation_ft": 79,
  "elevation_m": 24,
  "iata": "NSY",
  "icao": "LICZ",
  "latitude": 37.4017,
  "longitude": 14.9224,
  "name": "Sigonella Navy Air Base",
  "note": "C Di Palma",
  "reporting": true,
  "runways": [
    {
      "length_ft": 8077,
      "width_ft": 148,
      "surface": "asphalt",
      "lights": true,
      "ident1": "10R",
      "ident2": "28L",
      "bearing1": 98,
      "bearing2": 278
    },
    {
      "length_ft": 8012,
      "width_ft": 98,
      "surface": "asphalt",
      "lights": true,
      "ident1": "10L",
      "ident2": "28R",
      "bearing1": 98,
      "bearing2": 278
    }
  ],
  "state": "82",
  "type": "medium_airport",
  "website": null,
  "wiki": null
}
```


Figure 1 Sample Airport Station JSON data retrieved from AVWX Rest API for LIZC

The runway information contains bearing and runway identification number that are used. The METAR string or XML file containing wind direction can provide crosswind component information by using runway information. ∂ is the minimum of azimuth angle difference between runway directions and wind direction and crosswind speed can be calculated with formula below

$$Crosswindspeed = windspeed * sin(\partial)$$

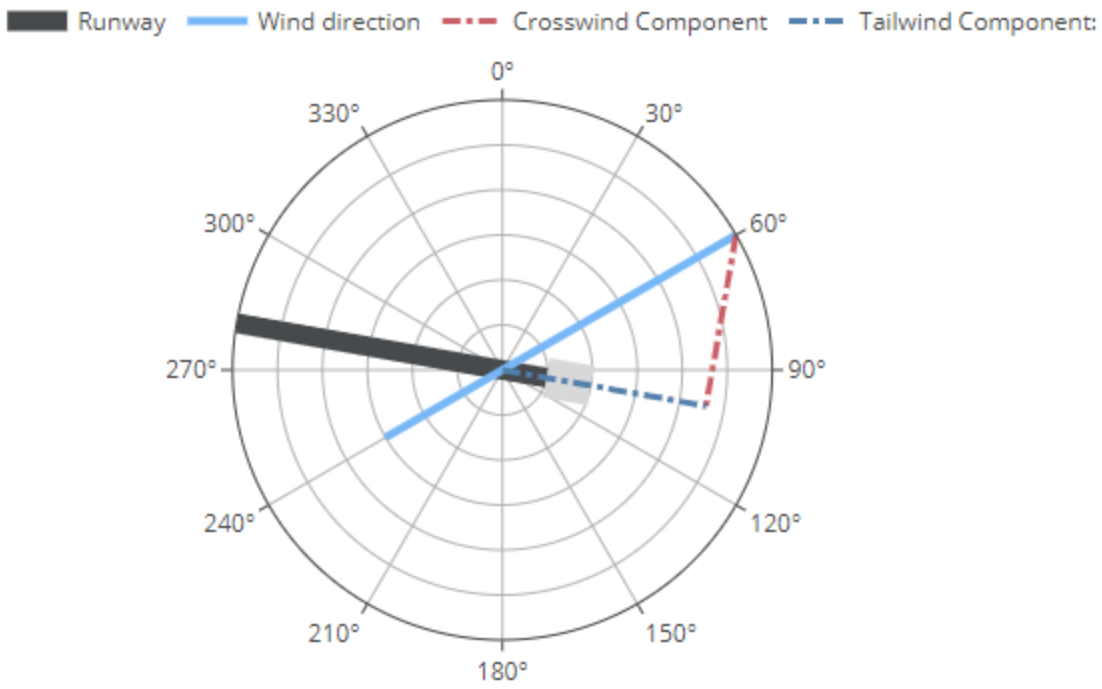


Figure 2 Cross wind representation with Runway 280 °, Wind direction 60 °, Wind Speed will result 3.8567 as crosswind component. (Figure retrieved from https://aerotoolbox.com/crosswind/)

CrossWind class has two methods, 1) calculateCrossWindComponent for both METAR raw string and XML format data 2) createModifiedIwxxmFileforCrossWind for XML format data.

displayMap method was created to visualize the wind direction and runway direction over folium map, that can only work in IPython (as Google Colab run in IPython) and not applicable for the console application.

Running Application with Python VirtualEnv

Generate virtual environment follow the link for instruction https://docs.python.org/3/tutorial/venv.html

Once you create a folder and activate your python virtual environment

1. Install required packages. **requirements.txt** file is provided and contains the necessary packages for the console application.

pip install -r requirements.txt

2. Install GIFTs. GIFTs can be downloaded manually or if GIT Tools (https://git-scm.com/) exists, git clone command can be used to download the files. Unzip the file and run setup.py file inside the project to install gifts package.

git clone https://github.com/NOAA-MDL/GIFTs.git

cd GIFTs

python setup.py install

3. Now you can run console application script with the given parameters. –h gives available options that can be used with examples

python MetocTools.py –h

```
MetocTools.py --XMLConversion --stationDesignator <ICOAStationDesign> --METARString <METARString>
     Example: MetocTools.py --XMLConversion --stationDesignator LICZ --METARString "260750Z 26010KT 9999 SCT028 BKN070 06/M01 Q1023 NOSIG RMK RWY24 27009KT="
MetocTools.py --XMLModification --inputXMLurl <URL> --ColorState --CrossWindAlert
     Example: MetocTools.py --XMLModification --inputXMLurl "http://www.meteocenter.ru/iwxxm/xml/A_LARU20UAKK130830_C_RUMS_20210413084323.xml" --ColorState --CrossWindAlert
MetocTools.py --XMLModification --inputXMLPath <Path> --ColorState --CrossWindAlert
     Example: MetocTools.py --XMLModification --inputXMLPath "/content/LICZ_IWXXM_File_XML_20210429085650.xml" --ColorState --CrossWindAlert
MetocTools.py --SearchMETARString --folderPathtoSearch <SearchFolderPath> --stationDesignator <stationDesignator>
     Example: MetocTools.py --SearchMETARString --folderPathtoSearch "/content/content/SmallFiles4" --stationDesignator SECU
MetocTools.py --SearchMETARString --folderPathtoSearch <SearchFolderPath> --stationDesignator <stationDesignator> --XMLConversion
     Example: MetocTools.py --SearchMETARString --folderPathtoSearch "/content/content/SmallFiles4" --stationDesignator SECU --XMLConversion
```

Figure 3 The –h option show available uses

4. Convert Metar String to XML file:
   -- **XMLConversion** option specify that given a Metar String will be converted to XML file, but has no input
   --**stationDesignator** option specify station designator that takes ICAO airport location designator as input
   -- **METARString** option takes the user given METAR message as input

python MetocTools.py --XMLConversion --stationDesignator LICZ --
METARString "260750Z 26010KT 9999 SCT028 BKN070 06/M01 Q1023 NOSIG RMK RWY24 27009KT="

```
LICZ
260750Z 26010KT 9999 SCT028 BKN070 06/M01 Q1023 NOSIG RMK RWY24 27009KT=
Processing  LICZ 260750Z 26010KT 9999 SCT028 BKN070 06/M01 Q1023 NOSIG RMK RWY24 27009KT=
...Gift Module exist, it will check for aerodromes.tbl and aerodromes.db

...Existing aerodromes.tbl and aerodromes.db

File in process: LICZ_IWXXM_File_XML_20210505120439.xml
File has been created: LICZ_IWXXM_File_XML_20210505120439.xml
```

Figure 4 Results of running command will show the name of created file for created XML

Figure 5 Resulting XML file can be viewed though browser

Conversion requires aerodromes database to be in same folder location as the running scripts. If it is not exist it will try to access http://ourairports.com/data/airports.csv to create tbl and db files.

5. Modify existing (IWXXM format) XML file with calculated ColorState and CrossWindAlert

   --XMLModification option specify that given xml file will be modified with calculated parameters

   --inputXMLPath option specify location of xml file as input

   --ColorState option specify to get color state calculation and has no input

   --CrossWindAlert option specify to get crosswind component calculation and has no input

There is also an option for url instead of file path to retrieve XML files, example can be seen with –h option

```
python MetocTools.py --XMLModification --
inputXMLPath "/content/LICZ_IWXXM_File_XML_20210505120439.xml" --
ColorState --CrossWindAlert
```

Figure 6 Results of running command will show the name of created file for modified XML

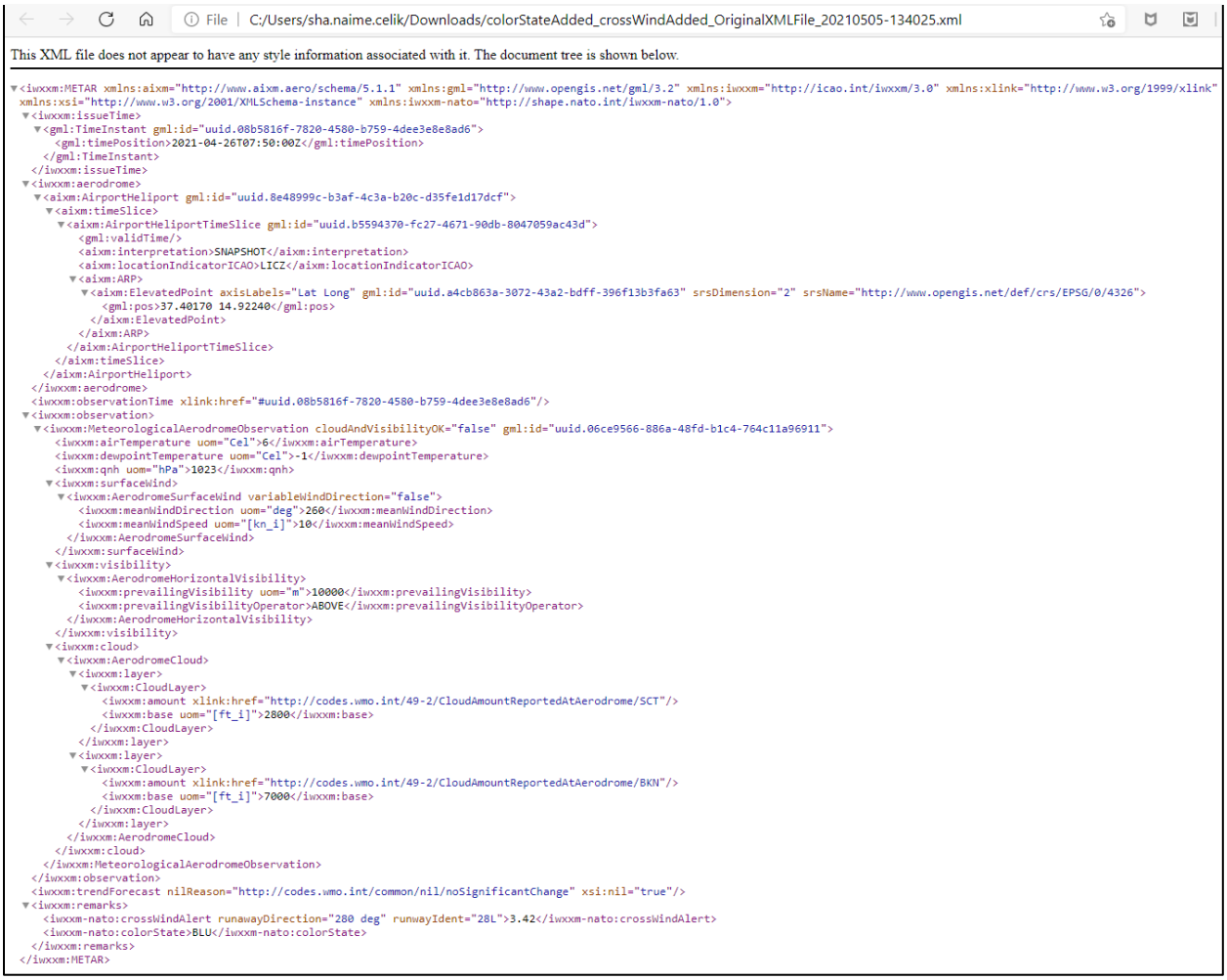


Figure 7 Modified XML file after calculation

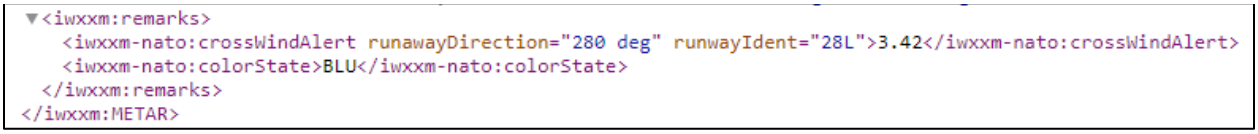


Figure 8 The included part of the xml parameters under remarks

6. Find most recent metarString from Bullletin Files.

   -- **SearchMETARString** specify that metar string with the recent date will be retrieved

   -- **folderPathtoSearch** option provides folder that will be search trough as an input

-- stationDesignator option specify the station designator for the search as input

python MetocTools.py --SearchMETARString --folderPathtoSearch "/content/sampleFiles" --stationDesignator CZOL

```
/content/content/SmallFiles4
CZOL
METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011

METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011

METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011

METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011

METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011

METAR CZOL 081500Z AUTO 34013KT 02/02 RMK AO1 PK WND 35020/1456

METAR CZOL 081500Z AUTO 34013KT 02/02 RMK AO1 PK WND 35020/1456

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

METAR CZOL 070000Z AUTO 11002KT 12/M05 RMK AO1 T01171048=

MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 090100Z 33007KT 06/M12 RMK AO1 PK WND 32017/0011', visibility=None
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 090100Z 33007KT 06/M12 RMK AO1 PK WND 32017/0011', visibility=None
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 090100Z 33007KT 06/M12 RMK AO1 PK WND 32017/0011', visibility=None
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 090100Z 33007KT 06/M12 RMK AO1 PK WND 32017/0011', visibility=None
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 090100Z 33007KT 06/M12 RMK AO1 PK WND 32017/0011', visibility=None
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 081500Z 34013KT 02/02 RMK AO1 PK WND 35020/1456', visibility=None,
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 081500Z 34013KT 02/02 RMK AO1 PK WND 35020/1456', visibility=None,
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
MetarData(altimeter=None, clouds=[], flight_rules='IFR', other=[], sanitized='CZOL 070000Z 11002KT 12/M05 RMK AO1 T01171048=', visibility=None, wind_
--------Most Recent Metar String----------
METAR CZOL 090100Z AUTO 33007KT 06/M12 RMK AO1 PK WND 32017/0011
```

Figure 9 Result of METAR string search for a given station designator

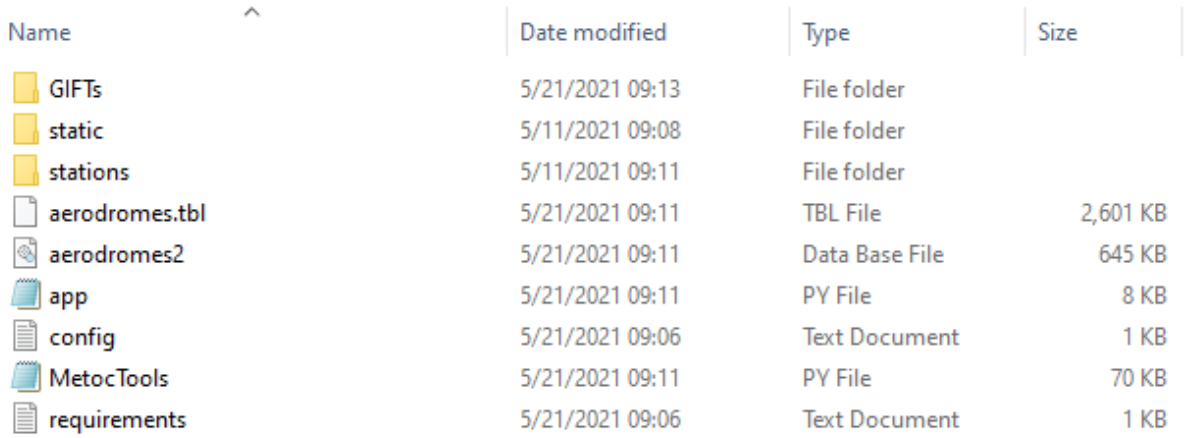
Running Swagger Web Application with Docker

The necessary instructions to run docker application can be seen on
https://docs.docker.com/get-docker/

```
FROM centos/python-38-centos7

## Source: https://hub.docker.com/r/gpmidi/centos-6.5

WORKDIR /tmp/src

ADD .  /tmp/src

USER root

RUN wget https://github.com/NOAA-MDL/GIFTs/archive/refs/heads/master.zip -O GIFTs.zip

RUN unzip GIFTs.zip

RUN cp -r GIFTs-master GIFTs

RUN rm -r GIFTs-master

RUN pip install -r requirements.txt

RUN cd GIFTs &&\

   python setup.py install &&\

   cd ..

CMD ["python", "app.py"]
```

Figure 10 The docker file to run the application in the working directory

The application in the local current directory contains; app.py for running swagger web application in the localhost, MetocTools.py, requirement.txt, static folder containing swagger.json, station folder containing .json files for stations and aerodromes for stations.



Name	Date modified	Type	Size
GIFTs	5/21/2021 09:13	File folder	
static	5/11/2021 09:08	File folder	
stations	5/11/2021 09:11	File folder	
aerodromes.tbl	5/21/2021 09:11	TBL File	2,601 KB
aerodromes2	5/21/2021 09:11	Data Base File	645 KB
app	5/21/2021 09:11	PY File	8 KB
config	5/21/2021 09:06	Text Document	1 KB
MetocTools	5/21/2021 09:11	PY File	70 KB
requirements	5/21/2021 09:06	Text Document	1 KB

Figure 11 The necessary files needs to be in same directory of the Dockerfile

Once the docker application working on the server side (in our case we will be using localhost). The Dockerfile containing the necessary commands can be used to build docker image for metoctools.

1. Build docker image for metoctools.

docker build -t metoctool .

2. Run docker container with port 5001

docker run -p 5001:5001 metoctool

The application will be running in 0.0.0.0:5001 or 127.0.0.1:5001 depending on swagger.json configuration. Use Firefox to test the running application.

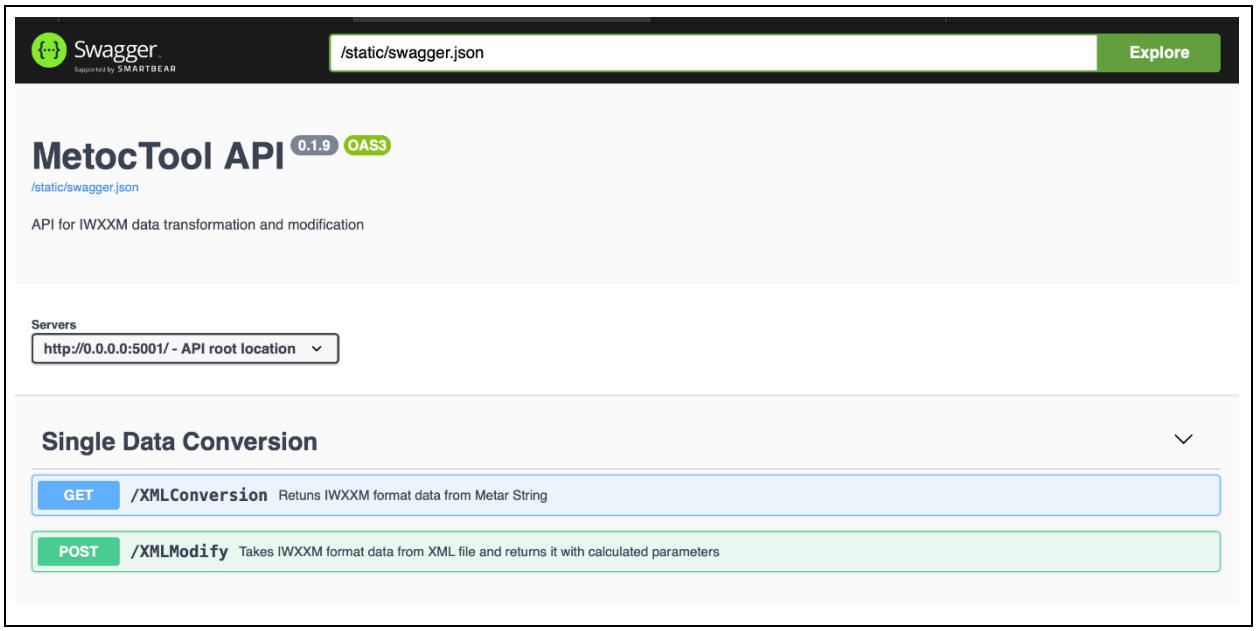


Figure 12 The swagger web service created for Metar Data conversion and XML Modification

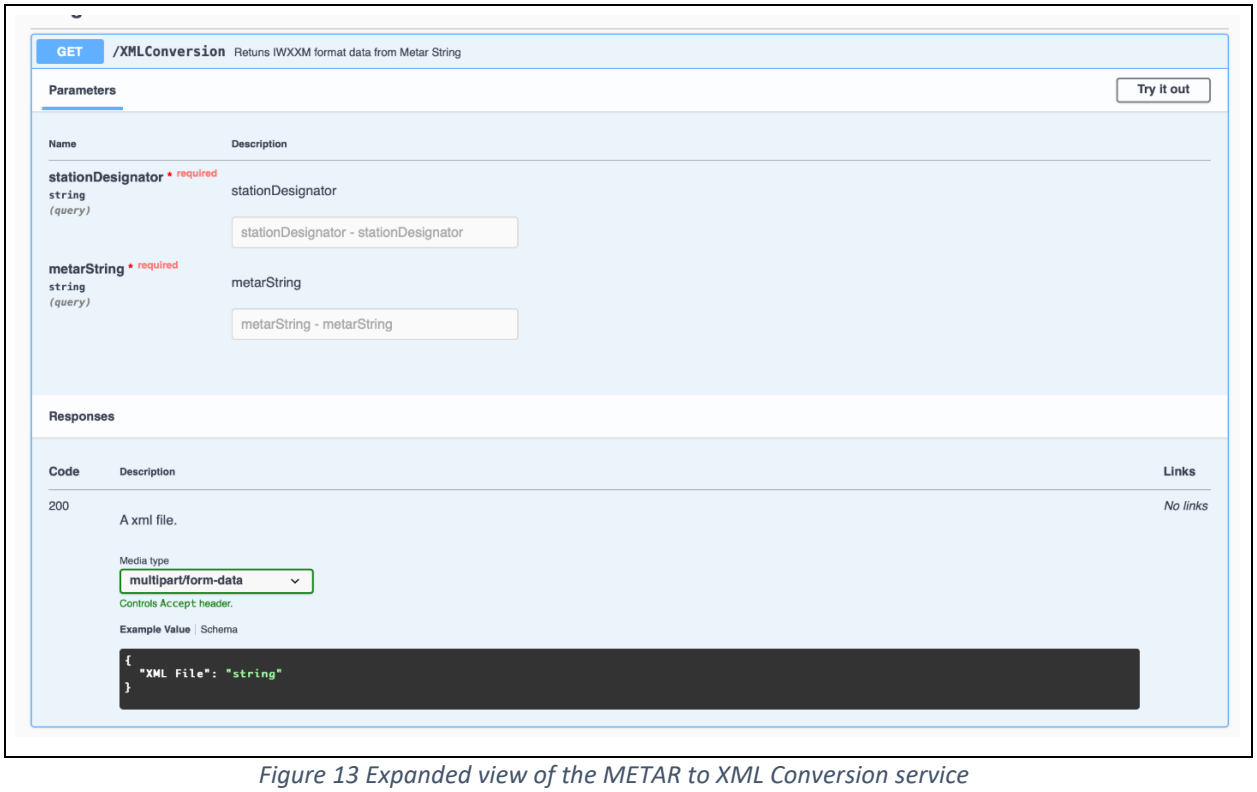


Figure 13 Expanded view of the METAR to XML Conversion service

1. To convert a user given METAR string to IWXXM format XML file.

After pressing try it out button, the user provides the station designator and METAR string to the application. Once the user requests data with execute button, the application returns the xml file within the response body that can be downloaded.

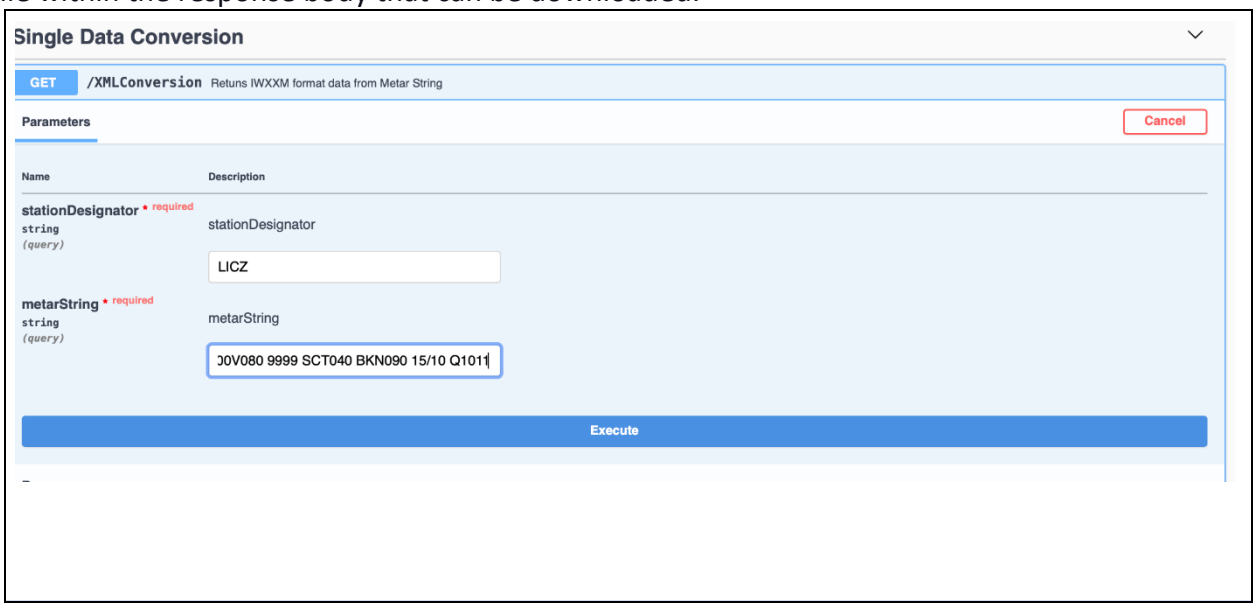


Figure 14 The view of the service method before execution

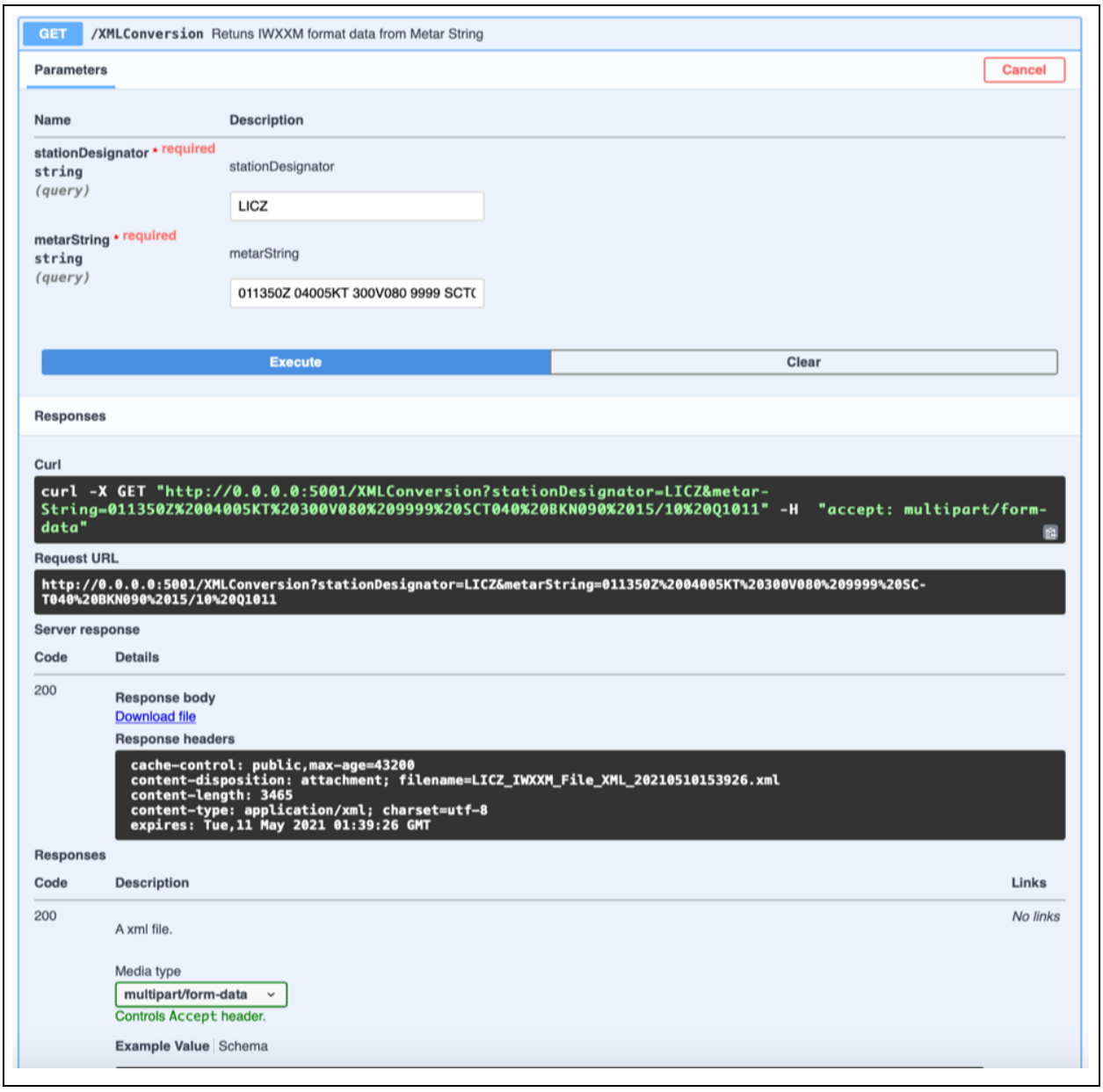


Figure 15 Response body will provide link to download converted XML file

Figure 16 Downloaded XML file from response body

2. To modify an existing XML document in IWXXM format and, add ColorState and CrossWindAlert with “iwxxm-nato” XML elements under remarks by providing a file given in a URL or a direct path.

Similarly, the user provides a IWXXM format XML file with the **choose file button** and indicate parameters that will be calculated. The response body will returned modified XML file to the user.

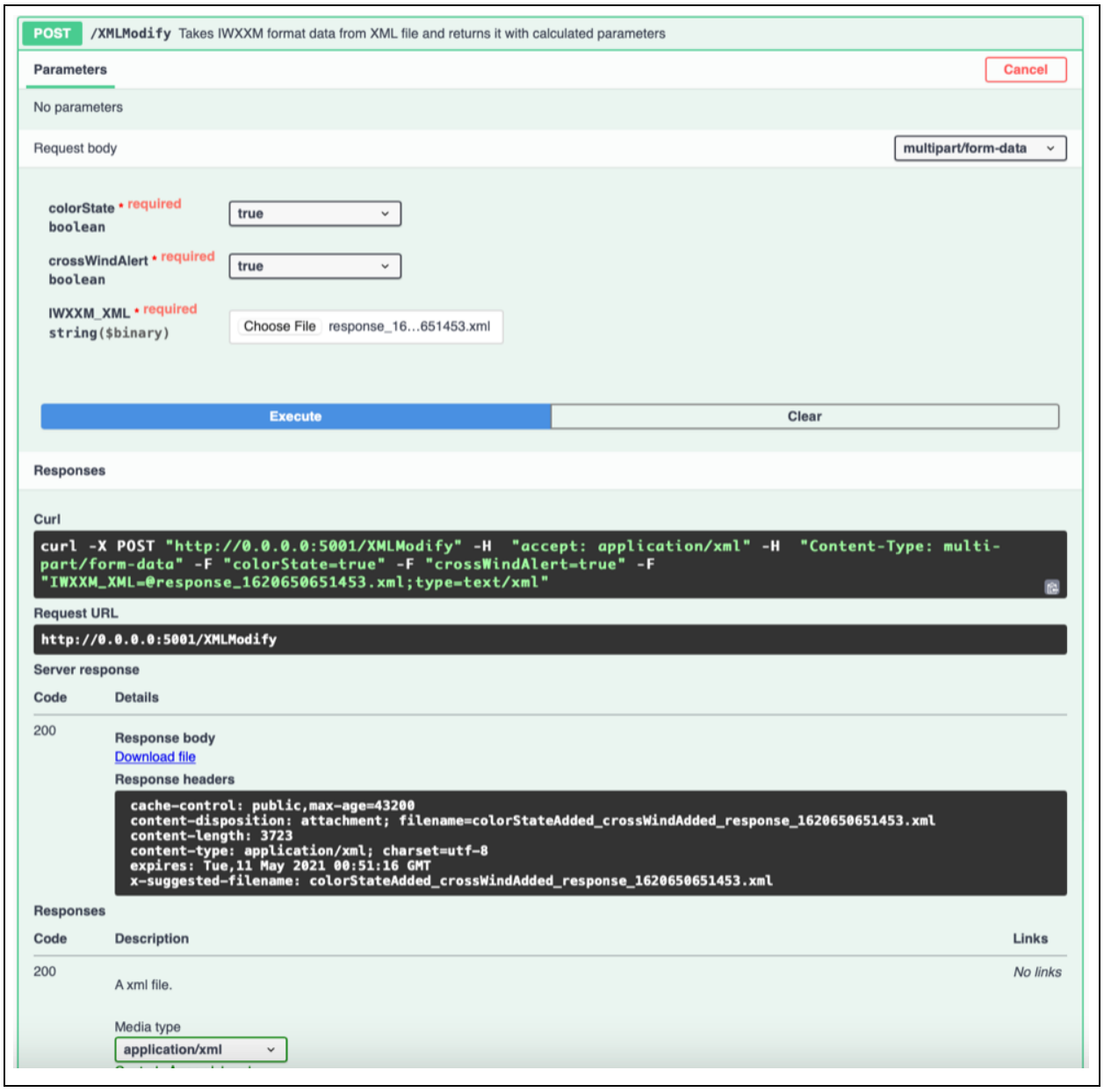
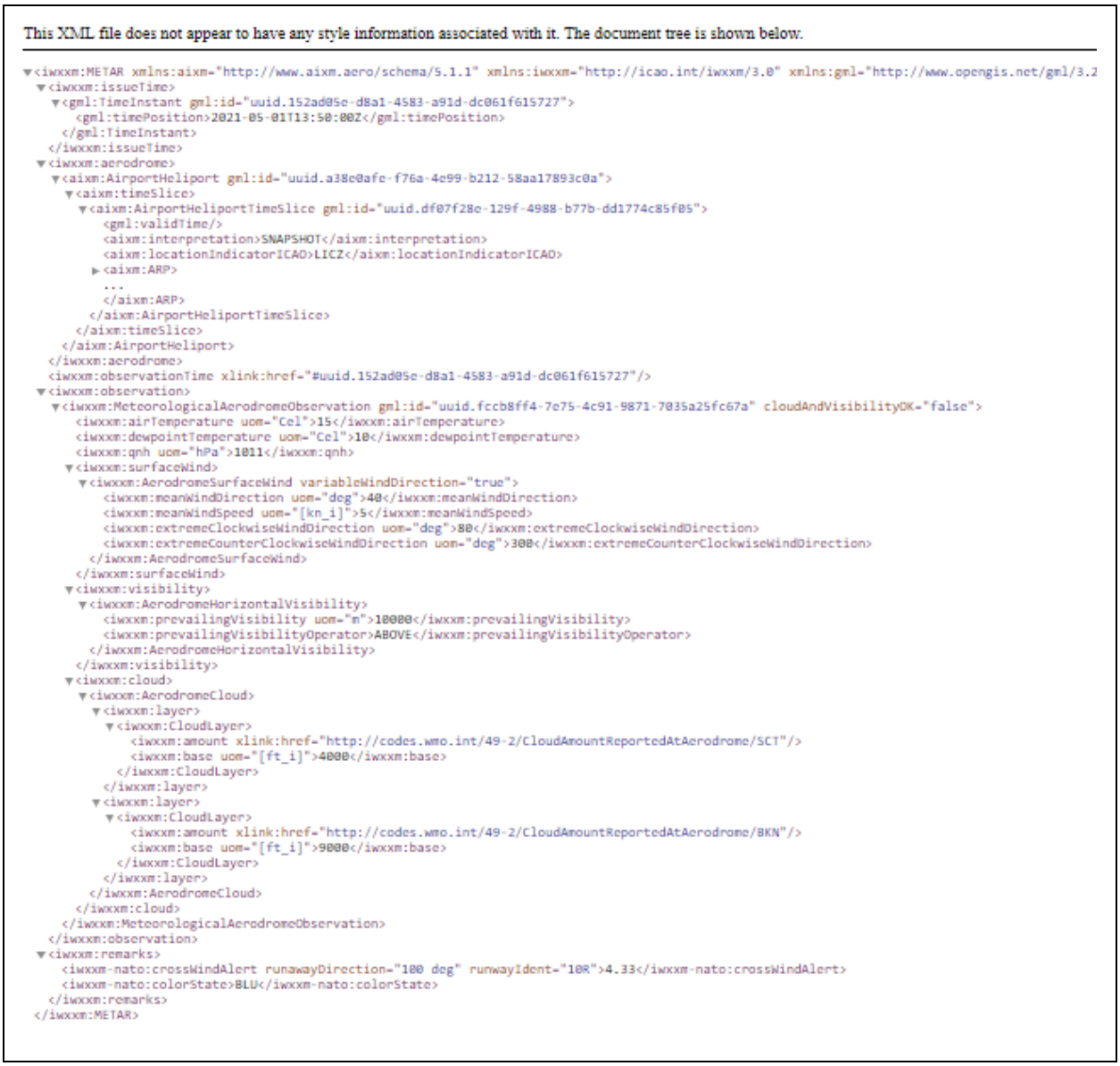


Figure 17 View from XMLModify service after execution

This XML file does not appear to have any style information associated with it. The document tree is shown below.




```
▼<iwxxm:METAR xmlns:aixm="http://www.aixm.aero/schema/5.1.1" xmlns:iwxxm="http://icao.int/iwxxm/3.0" xmlns:gml="http://www.opengis.net/gml/3.2
  ▼<iwxxm:issueTime>
    ▼<gml:TimeInstant gml:id="uuid.152ad05e-d8a1-4583-a91d-dc061f615727">
       <gml:timePosition>2021-05-01T13:50:00Z</gml:timePosition>
     </gml:TimeInstant>
   </iwxxm:issueTime>
  ▼<iwxxm:aerodrome>
    ▼<aixm:AirportHeliport gml:id="uuid.a38e0afe-f76a-4e99-b212-58aa17893c0a">
      ▼<aixm:timeSlice>
        ▼<aixm:AirportHeliportTimeSlice gml:id="uuid.df07f28e-129f-4988-b77b-dd1774c85f05">
           <gml:validTime/>
           <aixm:interpretation>SNAPSHOT</aixm:interpretation>
           <aixm:locationIndicatorICAO>LICZ</aixm:locationIndicatorICAO>
         ▶<aixm:ARP>
           ...
           </aixm:ARP>
         </aixm:AirportHeliportTimeSlice>
       </aixm:timeSlice>
     </aixm:AirportHeliport>
   </iwxxm:aerodrome>
   <iwxxm:observationTime xlink:href="#uuid.152ad05e-d8a1-4583-a91d-dc061f615727"/>
  ▼<iwxxm:observation>
    ▼<iwxxm:MeteorologicalAerodromeObservation gml:id="uuid.fccb8ff4-7e75-4c91-9871-7035a25fc67a" cloudAndVisibilityOK="false">
       <iwxxm:airTemperature uom="Cel">15</iwxxm:airTemperature>
       <iwxxm:dewpointTemperature uom="Cel">10</iwxxm:dewpointTemperature>
       <iwxxm:qnh uom="hPa">1011</iwxxm:qnh>
      ▼<iwxxm:surfaceWind>
        ▼<iwxxm:AerodromeSurfaceWind variableWindDirection="true">
           <iwxxm:meanWindDirection uom="deg">40</iwxxm:meanWindDirection>
           <iwxxm:meanWindSpeed uom="[kn_i]">5</iwxxm:meanWindSpeed>
           <iwxxm:extremeClockwiseWindDirection uom="deg">80</iwxxm:extremeClockwiseWindDirection>
           <iwxxm:extremeCounterClockwiseWindDirection uom="deg">300</iwxxm:extremeCounterClockwiseWindDirection>
         </iwxxm:AerodromeSurfaceWind>
       </iwxxm:surfaceWind>
      ▼<iwxxm:visibility>
        ▼<iwxxm:AerodromeHorizontalVisibility>
           <iwxxm:prevailingVisibility uom="m">10000</iwxxm:prevailingVisibility>
           <iwxxm:prevailingVisibilityOperator>ABOVE</iwxxm:prevailingVisibilityOperator>
         </iwxxm:AerodromeHorizontalVisibility>
       </iwxxm:visibility>
      ▼<iwxxm:cloud>
        ▼<iwxxm:AerodromeCloud>
          ▼<iwxxm:layer>
            ▼<iwxxm:CloudLayer>
               <iwxxm:amount xlink:href="http://codes.wmo.int/49-2/CloudAmountReportedAtAerodrome/SCT"/>
               <iwxxm:base uom="[ft_i]">4000</iwxxm:base>
             </iwxxm:CloudLayer>
           </iwxxm:layer>
          ▼<iwxxm:layer>
            ▼<iwxxm:CloudLayer>
               <iwxxm:amount xlink:href="http://codes.wmo.int/49-2/CloudAmountReportedAtAerodrome/BKN"/>
               <iwxxm:base uom="[ft_i]">9000</iwxxm:base>
             </iwxxm:CloudLayer>
           </iwxxm:layer>
         </iwxxm:AerodromeCloud>
       </iwxxm:cloud>
     </iwxxm:MeteorologicalAerodromeObservation>
   </iwxxm:observation>
  ▼<iwxxm:remarks>
     <iwxxm-nato:crossWindAlert runawayDirection="100 deg" runwayIdent="10R">4.33</iwxxm-nato:crossWindAlert>
     <iwxxm-nato:colorState>BLU</iwxxm-nato:colorState>
   </iwxxm:remarks>
 </iwxxm:METAR>
```


Figure 18 Downloaded XML file from response body

References

AVXW REST API. (2021). AVWX · Apiary. AVWX · Apiary. https://avwx.docs.apiary.io/#

Cx_freeze. (2021). *Cx-freeze*. PyPI. https://pypi.org/project/cx-Freeze/

Duarte M. (2021). *Frequently asked questions — cx_Freeze 6.6 documentation*. Welcome to cx_Freeze's documentation — cx_Freeze 6.6 documentation. https://cx-freeze.readthedocs.io/en/latest/faq.html#problems-with-running-frozen-programs

DuPont M. (2021). *Avwx-rest/avwx-engine*. GitHub. https://github.com/avwx-rest/avwx-engine

Oberfield M. (2021, January 12). *GIFTs:Generate IWXXM From TAC*. GitHub. https://github.com/NOAA-MDL/GIFTshttps://github.com/NOAA-MDL/GIFTs

Appendix

```
def createJsonFilesforStations (airpotDataUrl='http://ourairports.com/data/airports.csv'):

        df = pd.read_csv(airpotDataUrl)
        df.head()
        #print(df['ident'].to_string())
        ndf=pd.DataFrame()
        ndf['ICAO'] = df['ident']
        ndf['IATA'] = ""
        ndf['AltID'] = ""
        ndf['FullName'] = ""  ### The name removed
        ndf["Latitude"]=df["latitude_deg"]
        ndf["Longitude"]=df["longitude_deg"]
        ndf["Elevation"]=0
        # Field #1 = ICAO identifier - 4 characters (required)
        # Field #2 = IATA identifier - 3 characters (optional)
        # Field #3 = Alternate identifier 3-6 characters (optional)
        # Field #4 = Full name of aerodrome, up to 60 characters (optional)
        # Field #5 = Latitude of aerodrome in degrees (decimal)  (southern latitudes are negative) (required)
        # Field #6 = Longitude of aerodrome in degrees (decimal) (western longitudes are negative) (required)
        # Field #7 = Elevation of aerodrome in metres (required)

        folderPathForStationFiles="./stations"
        for i, row in df.iterrows():
         stationDesignator=row['ident']
         if len(stationDesignator)==4 and stationDesignator.isalpha():
            print (stationDesignator)
            filePathforStationFile=folderPathForStationFiles+"/"+stationDesignator+"_station.json"
            jsonResponseStation=getStationInfofromFile(filePathforStationFile)
            if jsonResponseStation==None:
             jsonResponseStation=getStationInfo(stationDesignator)
             try:

               if jsonResponseStation["runways"][0]["bearing2"]!=None:
                    writeStationInfoIntoFile(jsonResponseStation,filePathforStationFile)
                    print (jsonResponseStation["runways"])

             except:
              continue
createJsonFilesforStations()
```

Figure 19 Function to retrieve airport station json files containing runway information from AVWX Rest API

```
def createAirportTableforGIFTs (self,airpotDataUrl='http://ourairports.com/data/airports.csv', filePathforFile='./aerodromes.tbl'):


        df = pd.read_csv(airpotDataUrl)

        df.head()

        ndf=pd.DataFrame()

        ndf['ICAO'] = df['ident']

        ndf['IATA'] = ""

        ndf['AltID'] = ""

        ndf['FullName'] = ""  ### The name removed

        ndf["Latitude"]=df["latitude_deg"]

        ndf["Longitude"]=df["longitude_deg"]

        ndf["Elevation"]=0

        # Field #1 = ICAO identifier - 4 characters (required)
        # Field #2 = IATA identifier - 3 characters (optional)
        # Field #3 = Alternate identifier 3-6 characters (optional)
        # Field #4 = Full name of aerodrome, up to 60 characters (optional)
        # Field #5 = Latitude of aerodrome in degrees (decimal)  (southern latitudes are negative) (required)
        # Field #6 = Longitude of aerodrome in degrees (decimal) (western longitudes are negative) (required)
        # Field #7 = Elevation of aerodrome in metres (required)
        ndf.head()
        pd.options.display.float_format = '{:.6f}'.format
        x = ndf.to_string(header=False,
          index=False,
          index_names=False).split('\n')

        vals = ['|'.join(ele.split()) for ele in x]
        with open(filePathforFile,'w') as f:
           f.write(
              ndf.to_csv(sep="|",index=False,header=False)
           )
        print ("aerodromes.tbl has been created!")
        return filePathforFile
```

```
    def createAerodromesDBforGIFTs(self,filePathforFile='./aerodromes.tbl'):
            database = {}
            with open(filePathforFile) as _fh:
              for lne in _fh:
                if lne.startswith('#'):
                  continue
                try:
                  sid, IATAId, alternateId, name, lat, lon, elev = lne.split('|')
                except ValueError:
                  continue
                if len(sid) == 4 and sid.isalpha():
                  database[sid] = '%s|%s|%s|%.5f %.5f %d' % (name[:60].strip().upper(), IATAId[:3].strip().upper(), alternateId[:6].strip().upper(), float(lat),
float(lon), int(elev))
            with open('./aerodromes2.db', 'wb') as _fh:
              pickle.dump(database, _fh, protocol=pickle.HIGHEST_PROTOCOL)
```

Figure 20 Function to create Aerodromes tbl and db file

```
FROM centos/python-38-centos7

## Source: https://hub.docker.com/r/gpmidi/centos-6.5

WORKDIR /tmp/src
ADD .  /tmp/src
USER root

RUN wget http://nixos.org/releases/patchelf/patchelf-0.10/patchelf-0.10.tar.bz2
RUN tar xf patchelf-0.10.tar.bz2
RUN cd patchelf-0.10 &&\
  ./configure --prefix="$HOME/.local" &&\
   make install &&\
   strip --strip-unneeded ~/.local/bin/patchelf &&\
   gzip -9 ~/.local/share/man/man1/patchelf.1

RUN pip install cx_Freeze
RUN pip install numpy cython mdtraj
RUN pip install cffi setuptools wheel twine auditwheel
RUN pip uninstall sgp4
RUN pip install pip install sgp4

RUN wget https://github.com/NOAA-MDL/GIFTs/archive/refs/heads/master.zip -O GIFTs.zip
RUN unzip GIFTs.zip
RUN cp -r GIFTs-master GIFTs
RUN rm -r GIFTs-master
RUN pip install -r requirements.txt
RUN cd GIFTs &&\
   python setup.py install &&\
   cd ..
```

Figure 21 Docker file used for creating Docker container for testing cx_freeze for console application. (Unsuccessful due to segment fault error in Docker container for Linux)

```
from cx_Freeze import setup, Executable

# Dependencies are automatically detected, but some modules need help.
buildOptions = dict(
   packages = ['gifts'],
   excludes = [],
   # We can list binary includes here if our target environment is missing them.
  bin_includes = [
      ' libffi-devel '    ]
)

executables = [
   Executable(
      'MetocTools.py',
      base = None,
      targetName = 'sample-app',
   )
]

setup(
   name='Sample App',
   version = '0.1',
   description = 'Sample App',
   options = dict(build_exe = buildOptions),
   executables = executables
)
```

Figure 22 setup.py file to freeze python as executable (failed in Linux Docker container but works in windows –do not include bin files for windows and add the necessary packages in bin list for Linux

)

```
yum install gcc openssl-devel bzip2-devel libffi-devel zlib-devel
```

```
wget https://www.python.org/ftp/python/3.9.4/Python-3.9.4.tgz

tar xzf Python-3.9.4.tgz

cd Python-3.9.4

./configure --enable-optimizations

make altinstall
```

Figure 23 Commands to setup python from binary