08

**Fall**

**FSGIM Implementation Support**

**Contract No. SB1341-14-CQ-0043/ Order No.:16-443**



**NOAA Weather Retrieval And Conversion Application (NOAA Client)**

**Draft v1**

**April 19, 2018**

**Prepared by**

**ESTA International, LLC**

**2214 Rock Hill Road, Suite 180**

**Herndon, Virginia, 20170-4234, USA**



[www.ESTAInternational.com](http://www.ESTAInternational.com)

ESTA International, LLC has prepared this document under contract to NIST under contract number SB1341-14-CQ-0043/ order no.:16-443. Unless otherwise noted all information in this document is confidential and proprietary.

This work was completed through close cooperation between

NIST and ESTA International Project Team members.

NIST Project Team Members includes:

Dr. Martin J. Burns Project Technical Lead

Mr. Paul Boynton Project Manager

ESTA team members for this project include:

Mr. Nader Farah Project Manager

Mr. Allen J. Jones FSGIM Model Subject Matter Expert

Mr. Donald F. Coffin OpenESPI/ Green Button Subject Matter Expert

Mr. Abhinav Verma Software Developer

Please address all inquiries to:

Mr. Nader Farah

ESTA International, LLC

[nader.farah@ESTAinternational.com](mailto:nader.farah@ESTAinternational.com)

Table of Contents

[1 Overview 3](#_Toc512263405)

[1.1 Objective 3](#_Toc512263406)

[1.2 Application Output: XML Atom Syndication Format - RFC 4287 3](#_Toc512263407)

[1.3 Python Versioning and Interactive Development Environment 3](#_Toc512263408)

[2 Obtaining Weather Data 4](#_Toc512263409)

[2.1 Steps to obtain weather observation data 4](#_Toc512263410)

[2.2 Steps to obtain weather forecast data 5](#_Toc512263411)

[3 Application Structure 7](#_Toc512263412)

[3.1 Run 7](#_Toc512263413)

[3.1.1 Function: main(location, apiKey, requestType) 7](#_Toc512263414)

[3.1.2 Function: get\_data(property\_name, forecast\_data) 8](#_Toc512263415)

[3.2 Weather Report 8](#_Toc512263416)

[3.3 Weather Observation 8](#_Toc512263417)

[3.4 Weather Area 8](#_Toc512263418)

[3.5 Precipitation 8](#_Toc512263419)

[3.6 ChildElements and BaseElements 9](#_Toc512263420)

[4 External Dependencies 10](#_Toc512263421)

[4.1 Requests: HTTP for Humans 10](#_Toc512263422)

[4.1.1 Introduction 10](#_Toc512263423)

[4.1.2 Usage 10](#_Toc512263424)

[4.1.3 Steps to Install 10](#_Toc512263425)

[4.1.4 Version 10](#_Toc512263426)

[4.2 Dateutil 11](#_Toc512263427)

[4.2.1 Introduction 11](#_Toc512263428)

[4.2.2 Usage 11](#_Toc512263429)

[4.2.3 Steps to Install 11](#_Toc512263430)

[4.2.4 Version 11](#_Toc512263431)

[4.3 Oauth2 12](#_Toc512263432)

[4.3.1 Introduction 12](#_Toc512263433)

[4.3.2 Usage 12](#_Toc512263434)

[4.3.3 Steps to Install 12](#_Toc512263435)

[4.3.4 Version 12](#_Toc512263436)

[5 Setup 13](#_Toc512263437)

[5.1 Linux 13](#_Toc512263438)

[5.2 Windows 15](#_Toc512263439)

[6 References 17](#_Toc512264578)

Table of figures

[Figure 1: NOAA Client Overview Diagram 3](#_Toc512264579)

[Figure 2: Sequence Diagram: Get Weather Observation Data 4](#_Toc512264580)

[Figure 3: Sequence Diagram: Get Weather Forecast Data 5](#_Toc512264581)

[Figure 4: Snapshot of NOAA Client File Explorer 7](#_Toc512264582)

[Figure 5: Run.py file, Debug param 13](#_Toc512264583)

[Figure 6: Create and navigate into 'noaaclient' directory 13](#_Toc512264584)

[Figure 7: Create a virtual environment 13](#_Toc512264585)

[Figure 8: Install Requests 14](#_Toc512264586)

[Figure 9: Install 'dateutil' 14](#_Toc512264587)

[Figure 10: Install 'oauth2' 14](#_Toc512264588)

[Figure 11: Clone OpenFSGIM-NOAAClient repository 14](#_Toc512264589)

[Figure 12: Execute run.py file 14](#_Toc512264590)

[Figure 13: Complete Setup demonstration for Linux 15](#_Toc512264591)

[Figure 14: Windows: Install requests 15](#_Toc512264592)

[Figure 15: Windows: Install dateutil 16](#_Toc512264593)

[Figure 16: Windows: Install oauth2 16](#_Toc512264594)

# Overview

## Objective

The objective of NOAA Weather Retrieval and Conversion application (also called NOAA Client) is to provide a weather data feed to Facilities Smart Grid Information Model(FSGIM) Weather Web Server. The ‘NOAA Client’ extracts the weather data from the National Oceanic and Atmospheric Administration (NOAA) API endpoints. The weather data is mapped into FSGIM schema. After the successful mapping of the Geo-JSON weather data to the FSGIM schema, application generates an XML output as per the RFC 4287 standard. Please refer to Figure 1.

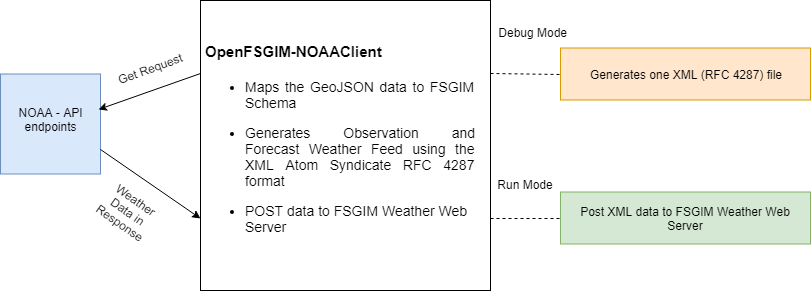


Figure : NOAA Client Overview Diagram

## Application Output: XML Atom Syndication Format - RFC 4287

The NOAA Client generates two types of weather data feeds: Historical observations and forecast. Both the feeds are generated in the form of XML Atom Syndication format [RFC 4287](https://tools.ietf.org/html/rfc4287)[[1]](#footnote-2).

## Python Versioning and Interactive Development Environment

The ‘NOAA Client’ uses Python 3.5[[2]](#footnote-3) and the development of this application has been completed using Pycharm Python IDE 2017.3.1 (Community Edition)[[3]](#footnote-4).

# Obtaining Weather Data

The following section explain the steps to obtain two types of weather data; forecast and observation. Please refer to the National Oceanic and Atmospheric Administration API documentation[[4]](#footnote-5).

## Steps to obtain weather observation data

The NOAA Client requests data from three end points in the NOAA API to extract weather observation data. The application uses Google Geocoding API to transform an address string into a location. A location consists of two values representing latitude and longitude.

The latitude and longitude values are used to obtain raw data provided by the Weather Office using the endpoint “/gridpoints/{wfo}/{x},{y}”. (Please refer to the NOAA API documentation to know more about the end points). The application extracts the observation stations URL by using the value stored in the Geo-JSON response field – ‘properties/ observationStations’. Please refer to Figure 2.

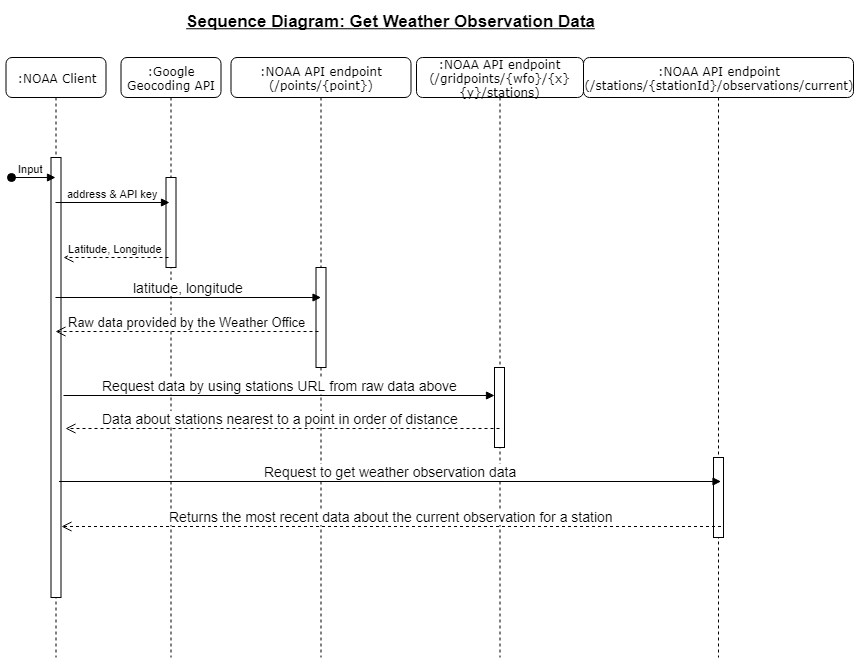


Figure : Sequence Diagram: Get Weather Observation Data

In the next step, the application uses the observation station URL from the last step to obtain observation station data. The station ID value is referenced by checking the first field value (‘properties/ stationIdentifier’) in features in the Geo-JSON response. The first value in features represent the closest station to the location specified by the latitude and longitude. The station ID is used to obtain weather observation data from the endpoint ‘/stations/{stationId}/observations/current’.

## Steps to obtain weather forecast data

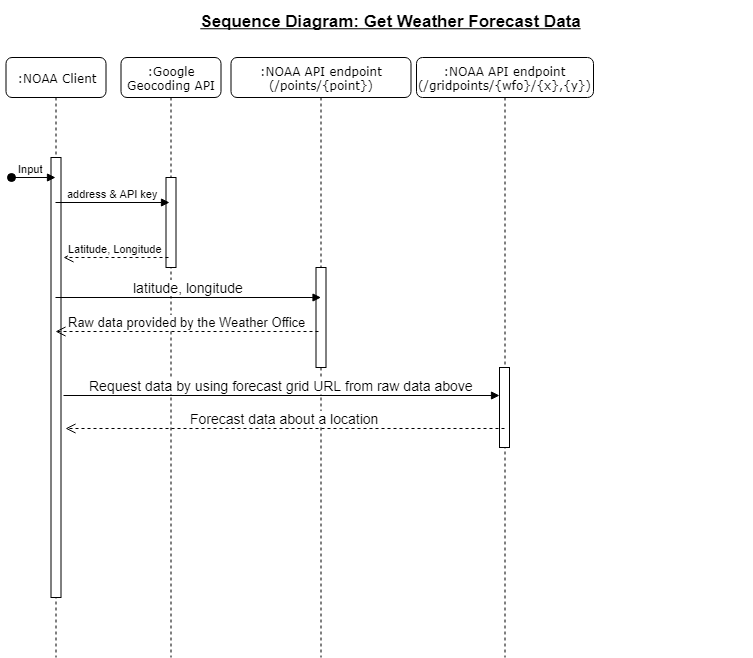


Figure : Sequence Diagram: Get Weather Forecast Data

The NOAA Client requests data from two NOAA API end points and one Google geocoding API endpoint. Just like for obtaining weather observation data the application uses Google geocoding API to transform an address string to a location. The location has two values representing latitude and longitude. These values are then used obtain raw data from the NOAA API endpoint “/gridpoints/{wfo}/{x},{y}”. The value of Forecast grid data URL is extracted from this data. This field value is present in the parameter (properties/forecastGridData) in the Geo-JSON raw data response received from the NOAA endpoint. Refer to Figure 3. The forecast grid data URL represents another NOAA API endpoint “/points/{point}”. This endpoint returns forecast information for a location.

# 

# Application Structure

Please refer to Figure 4 for complete list of files in the NOAA Client application.

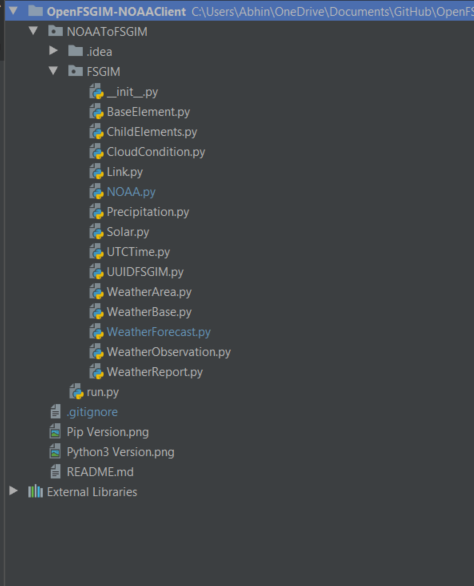


Figure : Snapshot of NOAA Client File Explorer

## Run

This file is at the center of this application. The responsibilities of the important functions of this file are discussed below:

### Function: main(location, apiKey, requestType)

The ‘main’ function orchestrates the calling of all the other functions in this application. This function is responsible for carrying out all the activities of this application in a sequential manner. The functionalities are completed in the following sequence:

1. ‘location’ parameter string in the function arguments is converted to a location (latitude and longitude). This operation is executed by ‘coordinatesFromAddress(location, apiKey)’ function.
2. Based on the location returned by the above step the weather data is collected. There are two functions in the run.py file which collect and transform the weather data based on the type of weather date requested (forecast or observation). The function which handles the weather forecast request is ‘add\_forecast\_data(feed\_xml, forecast\_data)’. The function which handles the weather observation request is ‘addWeatherObservation(feedxml, observationData)’. These two functions add the FSGIM mapped fields in a sequential order based on the FSGIM Schema.
3. The two functions introduced in the preceding steps create instances of ‘WeatherArea’, ‘WeatherBase’, ‘WeatherForecast’, ‘WeatherObservation’, ‘WeatherReport’ and ‘Precipitation’. The instances are created by calling the respective constructor of the class and adding the mapped value of the weather data as arguments to the constructors.
4. The instances generated from the individual components of ‘WeatherBase’, ‘WeatherForecast’, ‘WeatherObservation’, ‘WeatherReport’ and ‘Precipitation’ are grouped together on the basis of ‘mRID’ values.
5. After creating all the instances, a loop iterates over all the instances and generates XML entries for all the component stored as instances by invoking ‘getAtomEntry’ function which exists in every weather component class.
6. The XML feed created at this moment can be sent as a POST request to the FSGIM Weather Web Server or stored as a file on the local machine. The application decides between the two choices based on the Boolean variable present in this run.py file, called ‘debug’. This variable provides an additional flexibility in the system to analyze the XML atom file generated by this application. If this variable is set to TRUE than the application will store the XML feed as a file on the local machine otherwise it will be posted to the FSGIM Weather Web Server.

### Function: get\_data(property\_name, forecast\_data)

This function solves a very important objective of safe retrieval of property values from Python nested dictionary data structure. The Geo-JSON data received from the NOAA API is present as nested dictionaries inside this application. Dictionaries are very useful data structure and consist of two parameters; key and value. In terms of the weather response received from the NOAA API, the key represents the properties in the Geo-JSON response. This function was created to cover the situation of a missing property in the response. The ideal response of this application in case a property is missing is to report the missing property and continue extracting the available data from the NOAA response.

## Weather Report

This class is responsible for creating a weather report component entry inside the output XML file.

## Weather Observation

This class is responsible for creating a weather observation component entry inside the output XML file.

## Weather Area

This class is responsible for creating a weather area component entry inside the output XML file.

## Precipitation

This class is responsible for creating a precipitation entry inside the output XML file.

## ChildElements and BaseElements

These files contain reusable code for XML generation. This reusable code has been segregated into separate files to avoid repeatable code present in different parts of this application. These classes contain static functions and therefore to use these function, initialization of these classes is not required. By avoiding the initialization of these classes, the application reduces the temporary space required to execute this application in the host machine.

# External Dependencies

## Requests: HTTP for Humans[[5]](#footnote-6)

### Introduction

This library simplifies HTTP request and response communication. It is available on Python Package Index (PYPI).

### Usage

The ‘OpenFSGIM-NOAA\_Client’ uses this library to communicate with the National Oceanic and Atmospheric Administration (NOAA) application programming interface (API) to extract weather data.

The request library is used in the NOAA.py file inside this application. NOAA.py file creates a NOAA class which has three static functions. These functions are invoked by the run.py file in a synchronized manner to retrieve the weather data from the NOAA API. Please look at section 2 of this document for details.

### Steps to Install

The ‘requests’ library is easy to install. The library can be installed by the following command:

$pip install requests

### Version

‘OpenFSGIM-NOAAClient’ application uses 2.18.4 version of the requests library.

## Dateutil

### Introduction

The ‘dateutil’[[6]](#footnote-7) library is an extension to the standard datetime module, available in Python.

It is available on PYPI.

### Usage

The ‘OpenFSGIM-NOAA\_Client’ uses only a single module of the dateutil library called the parser module.

The parser module is used in two classes: ‘WeatherForecast’ and ‘WeatherObservation’. Both the classes use the parser module to convert date time value present in the Geo-JSON string response received from NOAA into a date time data type.

After conversion of the date time string value to a date time date type, the arithmetic operators can be used on the value to mold them into formats specified in the FSGIM schema.

### Steps to Install

The library can be installed by using the following command:

$pip install python-dateutil

### Version

The ‘OpenFSGIM-NOAAClient’ uses version 2.6.1 of the dateutil library

## Oauth2

### Introduction

The ‘Oauth2’[[7]](#footnote-8) library is a library used to provide Oauth authentication as per RF 5849 standard. It is available on PYPI.

### Usage

The ‘OpenFSGIM-NOAA\_Client’ uses ‘Oauth2’ library to post XML data to the FSGIM Weather Web server. The ‘OpenFSGIM-NOAA\_Client’ will be registered application on the FSGIM Weather Web Server and will be issued a key and a secret. This key and the secret will be hard coded in the run.py file in the ‘post\_data(post\_url, data)’ function.

The ‘Oauth2’ library is used in the ‘post\_data(post\_url, data)’ function.

This library is accessed by only a single file in this application called run.py file.

### Steps to Install

The library can be installed by using the following command:

$pip install python-oauth2

### Version

The ‘OpenFSGIM-NOAAClient’ uses version 1.1.0 of the ‘Oauth2’ library

# Setup

This section explains the steps to install and run the NOAA client.

## Linux

The NOAA client can work in two modes: to generate an XML file (called the Debug mode) and to post XML data on the FSGIM Weather Web Server (called the Run mode).

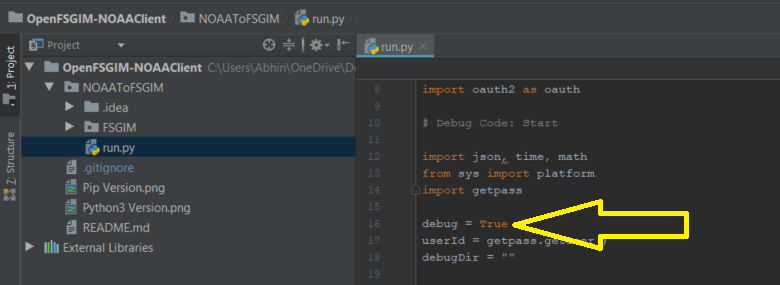


Figure : Run.py file, Debug param

In the production environment, this variable will be set as false to run the application in the Run mode.

Please follow the steps below to run this application on Linux:

1. Create a new directory ‘noaaclient’
2. Navigate into the new directory



Figure : Create and navigate into 'noaaclient' directory

1. Create a virtual environment[[8]](#footnote-9)

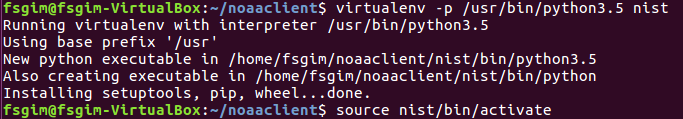


Figure : Create a virtual environment

1. Install requests

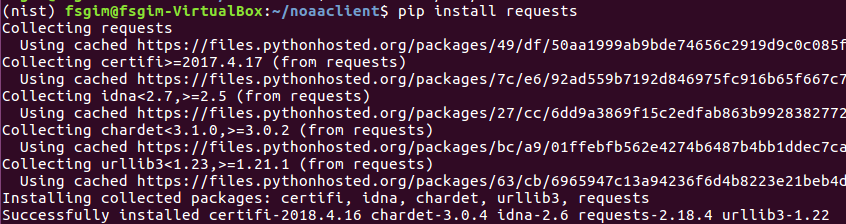


Figure : Install Requests

1. Install dateutil

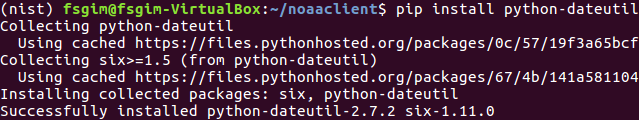


Figure : Install 'dateutil'

1. Install oauth2

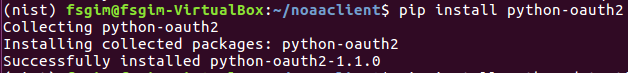


Figure : Install 'oauth2'

1. Clone or download git repository

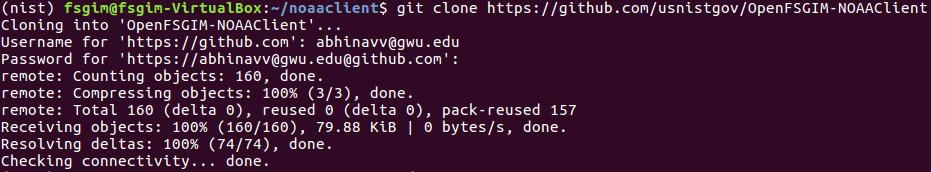


Figure : Clone OpenFSGIM-NOAAClient repository

1. Navigate to ‘NOAAToFSGIM’ directory
2. Execute run.py with the required parameters as shown





Figure : Execute run.py file

* 1. The run statement accepts the following three parameters:
     1. API Key: The API key can be created by using Google Geocoding API Developer Console[[9]](#footnote-10)
     2. Address: Enter the address of the location
     3. Observation Type: NOAA Client accepts two keywords: ‘observation’ / ‘forecast’

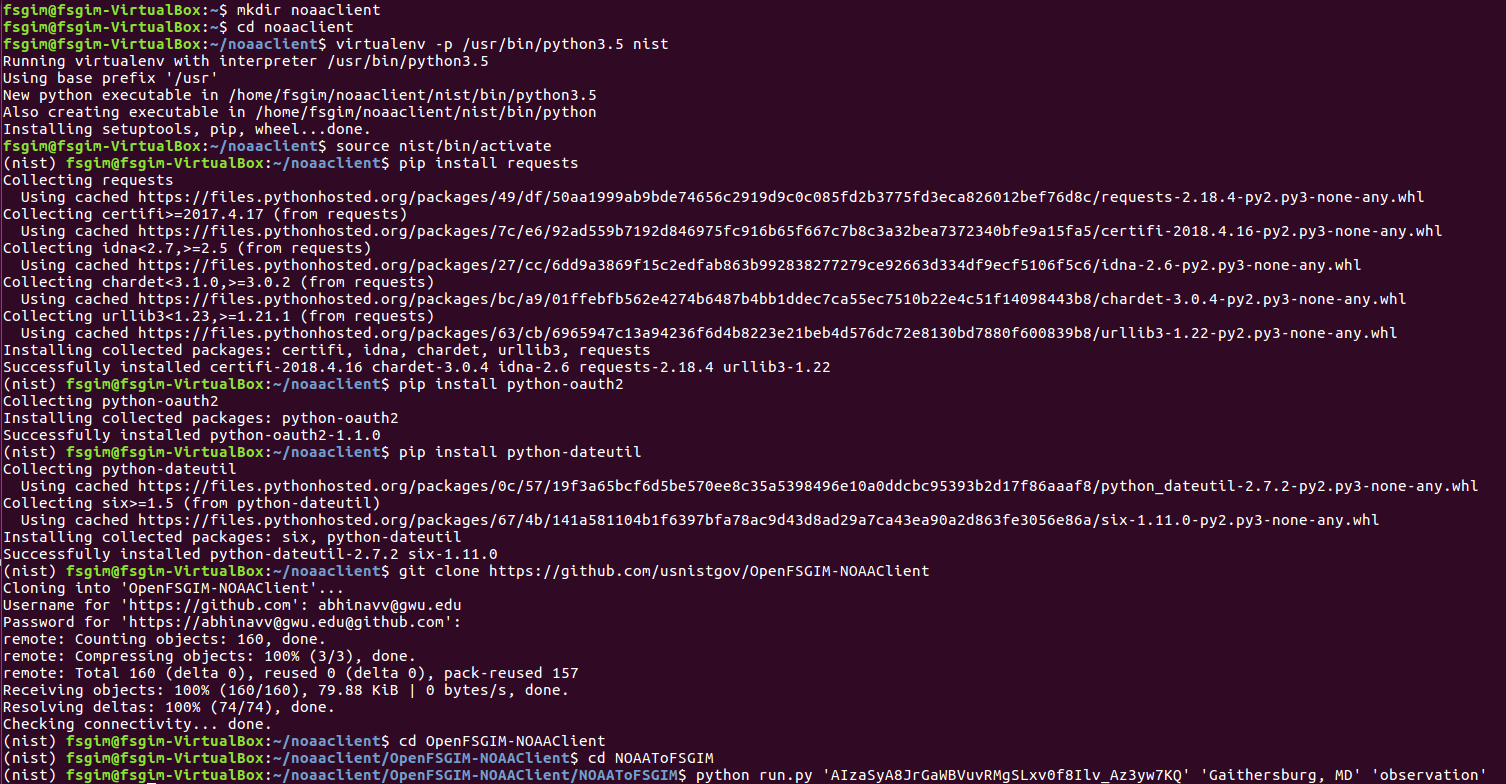


Figure : Complete Setup demonstration for Linux

## Windows

The following steps defines the setup of the NOAA Client on a windows machine:

1. Install requests

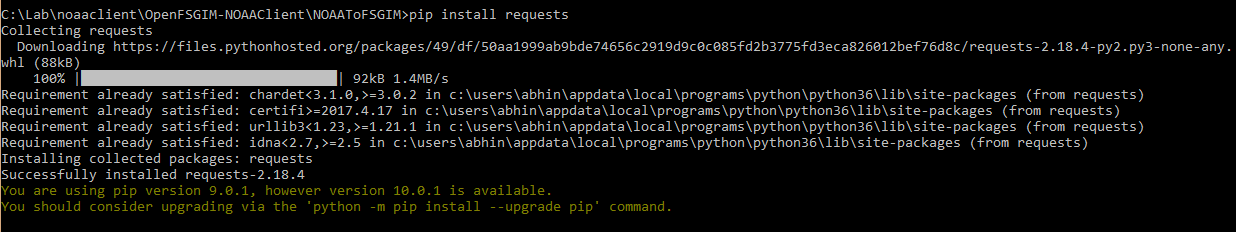


Figure : Windows: Install requests

1. Install python-dateutil

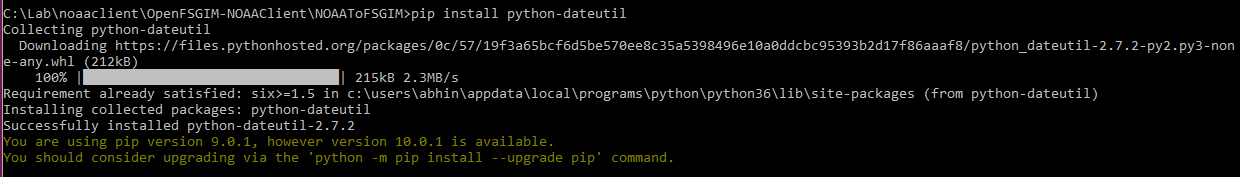


Figure : Windows: Install dateutil

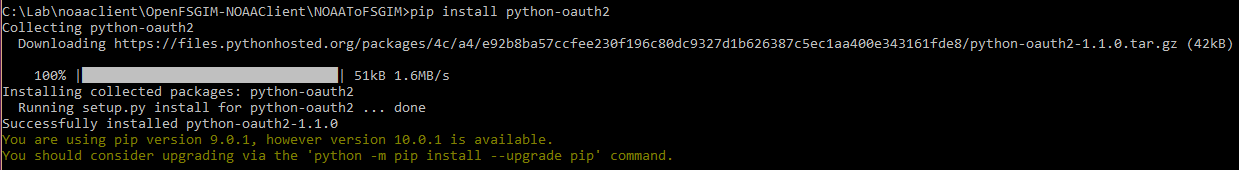
1. Install python-oauth2

Figure : Windows: Install oauth2

1. Follow the steps 7-9 from Section 5.1 above.

# References

|  |  |
| --- | --- |
| **Resource** | **Link** |
| RFC 4287 | <https://tools.ietf.org/html/rfc4287> |
| Python 3.5 | <https://www.python.org/downloads/release/python-350/> |
| Pycharm Python IDE 2017.3.1 (Community Edition) | <https://www.jetbrains.com/pycharm/download/#section=windows> |
| Python Virtual Environment Documentation | <https://docs.python.org/3/tutorial/venv.html> |
| National Oceanic and Atmospheric Administration API documentation | <https://forecast-v3.weather.gov/documentation?redirect=legacy> |
| Requests: HTTP for Humans | <http://docs.python-requests.org/en/master/> |
| Dateutil | <https://pypi.org/project/python-dateutil/> |
| Oauth2 | <https://pypi.org/project/python-oauth2/> |
| Google Geocoding API Developer Console | <https://console.developers.google.com/apis> |

1. https://tools.ietf.org/html/rfc4287 [↑](#footnote-ref-2)
2. https://www.python.org/downloads/release/python-350/ [↑](#footnote-ref-3)
3. https://www.jetbrains.com/pycharm/download/#section=windows [↑](#footnote-ref-4)
4. https://forecast-v3.weather.gov/documentation?redirect=legacy [↑](#footnote-ref-5)
5. http://docs.python-requests.org/en/master/ [↑](#footnote-ref-6)
6. https://pypi.org/project/python-dateutil/ [↑](#footnote-ref-7)
7. https://pypi.org/project/python-oauth2/ [↑](#footnote-ref-8)
8. https://docs.python.org/3/tutorial/venv.html [↑](#footnote-ref-9)
9. https://console.developers.google.com/apis [↑](#footnote-ref-10)