Visible Infrared Imaging Radiometer Suite (VIIRS) Enterprise Aerosol Detection Product (ADP) Users' Guide

Version 1, September 2020

Table of Contents

List of Figures

List of Tables

1. Purpose of this Guide

This VIIRS Enterprise Aerosol Detection Product (ADP) Environmental Data Record (EDR) User's Guide is intended for users of the Enterprise Processing System (EPS) version of the ADP EDRs generated from the Visible Infrared Imaging Radiometer Suite (VIIRS) on board the Suomi National Polar-Orbiting Partnership (SNPP) and the Joint Polar Satellite System (JPSS) satellites. It provides a general introduction to the VIIRS instrument, aerosol detection data products (including smoke/dust detection), format, and contents. It serves as an introduction and reference to the Algorithm Theoretical Basis Document (ATBD), a more detailed technical document describing the VIIRS Enterprise ADP algorithm in detail (see Section 5).

2. Points of Contact

For questions or comments regarding this document, please contact Shobha Kondragunta (Shobha.Kondragunta@noaa.gov) and Pubu Ciren (Pubu.Ciren@noaa.gov).

3. Document Definitions

The aerosol detection product (ADP) indicates the presence of dust or smoke aerosols above a threshold amount. Table 1 lists additional acronyms and abbreviations used in this document.

Table 1. List of acronyms and abbreviations used in this document.

Acronym/ Abbreviation	Definition
ADP	Aerosol Detection Product
ATBD	Algorithm Theoretical Basis Document
AVHRR	Advanced Very High Resolution Radiometer
CALIPSO	Cloud-Aerosol Lidar and Infrared Pathfinder Satellite Observation
CLASS	Comprehensive Large Array-Data Stewardship System
EDR	Environmental Data Record
EPS	Enterprise Processing System
IDL	Interactive Data Language
JPSS	Joint Polar Satellite System
MODIS	Moderate Resolution Imaging Spectroradiometer
RGB	True Color Imagery (Red-Green-Blue)
SNPP	Suomi National Polar-orbiting Partnership
VIIRS	Visible Infrared Imaging Radiometer Suite

4. VIIRS Overview

VIIRS is one of five instruments on board the SNPP and JPSS satellites. It is a scanning radiometer with capabilities that are intended to extend and improve upon or continue the heritage of AVHRR and MODIS. VIIRS data are used to measure cloud and aerosol properties, ocean color, sea and land surface temperature, ice motion and temperature, fires, and Earth's albedo. The SNPP and JPSS satellites are on an 824 km sun-synchronous orbit (inclination = 98.7°) with a 1:30 pm ascending

node. From this orbit, they achieve global coverage every day and have a repeat cycle of approximately 16 days. VIIRS has a swath width of 3,040 km with a spatial resolution of ~375 m at nadir in the Imagery (I) Bands and ~750 m at nadir in the Moderate (M) Bands. Through a system of pixel aggregation techniques, VIIRS controls pixel growth towards the edge of the scan such that the pixel sizes are comparable to nadir. For more information about this "bow-tie removal" aggregation scheme, consult the SDR User's Guide.

5. VIIRS Enterprise ADP Algorithm

The VIIRS ADP algorithm uses spectral and spatial threshold tests to identify pixels with smoke or dust aerosols. The algorithm treats detection differently for water and land. The current version of the algorithm does not detect aerosols in cloud-affected pixels, over snow/ice surface, or at night. Unlike VIIRS AOD, ADP is detected over sun glint water. The dust ADP over sun glint water should not be used, however, because of the high amount of false alarms.

The ADP Enterprise algorithm can run through three paths, depending on the spectral range of the input data: (1) IR-visible based detection, (2) deep-blue based detection, and (3) combined IR-visible based and deep-based detection. VIIRS spectral bands allow for option 3, the combined IR-visible based and deep-blue based detection. The deep-blue path permits assessment of the relative intensity of detected smoke and dust aerosols, while the IR-visible path indicates only the presence (yes/no) of smoke and dust aerosols. The VIIRS bands used as inputs to the ADP algorithm as listed in Table 2. The algorithm also utilizes upstream VIIRS products, including the VIIRS cloud mask and VIIRS snow/ice mask. Much more detail about the Enterprise ADP algorithm is given in the ATBD.

The ADP measurement range is set by a threshold of 0.2 AOD, which defines background atmospheric aerosol. Based on this measurement range, ADP accuracy is set as 80% of correct classification for dust over water and land, 80% of correct classification for smoke over land, and 70% correct classification for smoke over water. Verification of VIIRS ADP accuracy requirements is computed based on comparisons with true color (RGB) images, AERONET observations, and other satellite products such as CALIPSO, as described in the ATBD. The ADP algorithm performs most accurately for heavy smoke and dust episodes over dark surfaces. Smoke detection over semi-arid and arid regions is less accurate due to the lower contrast with the relatively bright surface.

6. VIIRS ADP Data Files

VIIRS ADP data are available on a granule basis and distributed in NetCDF4 format. Each granule has 48 scan lines (approximately 86 seconds), and the pixel-level M-band resolution (750 m at nadir) data is contained in 768 x 3200 arrays. Due to the relatively short granule length and the data resolution, users should expect a total of approximately 550 granule files for global coverage in the daytime each day.

Figure 1 breaks down the file naming convention for the VIIRS EPS ADP data files. The "satellite" is abbreviated "npp" for SNPP and "j01" for NOAA-20.

Table 2. List of VIIRS bands used in the ADP algorithms, specified by use as inputs for discriminating smoke, dust, or smoke and dust aerosols, as well as use in the deep-blue, IR-visible, or deep-blue and IR-visible algorithms.

VIIRS Band	Central Wavelength	Use in ADP Algorithm	
		Aerosol Type	Algorithm Path
M1	0.412µm	Dust and Smoke	Deep-Blue
M2	0.445 µm	Dust and Smoke	Deep-Blue
M3	0.488 µm	Dust and Smoke	Deep-Blue and IR-Visible
M4	0.555 µm	Smoke	Deep-Blue
M5	0.640 µm	Dust and Smoke	Deep-Blue and IR-Visible
M6	0.746 µm	Smoke	Deep-Blue
M7	0.865 µm	Dust and Smoke	Deep-Blue and IR-Visible
M8	1.24 µm	Dust and Smoke	Deep-Blue
M9	1.38 µm	Dust	IR-Visible
M10	1.61 µm	Smoke	IR-Visible
M11	2.25 µm	Dust and Smoke	Deep-Blue and IR-Visible
M12	3.70 µm	Dust and Smoke	IR-Visible
M13	4.05 µm	Smoke	IR-Visible
M15	10.7 µm	Dust and Smoke	IR-Visible
M16	12.01 µm	Dust	IR-Visible

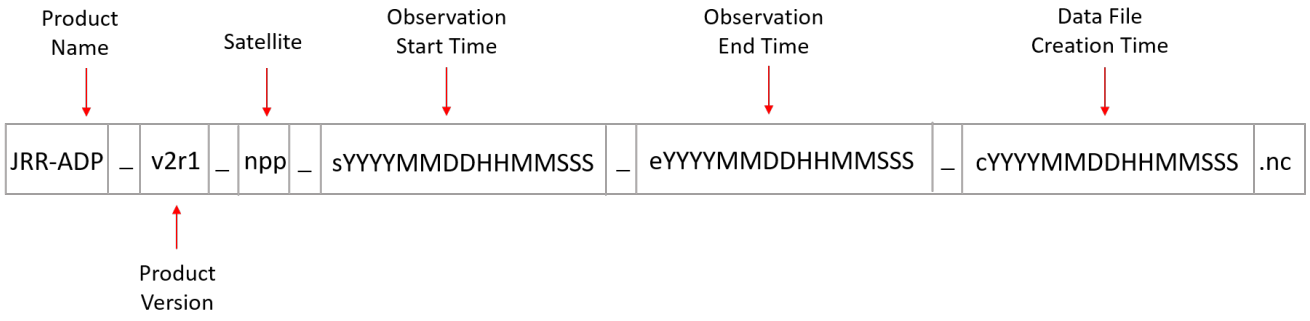

Figure 1. Enterprise VIIRS ADP filename definition and example.

The "product version" in the filename (e.g., "v2r1" in Figure 1) refers to the version of the entire EPS system, not the version of ADP data or the algorithm. Users should be aware of one major change in the VIIRS ADP product version history that impacts the contents of the data file: beginning at 00:00 UTC on August 13, 2018, the product version changed from v1r1 to v1r2. **Many of the VIIRS ADP variable names and descriptions changed with the transition from v1r1 to v1r2.** Users who work with VIIRS ADP data from prior to August 13, 2018 need to pay close attention to the discontinuity between key variable names and descriptions, as described in Section 7.

7. Working with VIIRS ADP Data

The VIIRS EPS ADP data file contains many variables, listed in Table 3. Users should be aware that many of the ADP variable names and definitions changed with the transition from product version v1r1 to v1r2 on August 13, 2018. In this section, variable names and definitions that are valid from August 13, 2018 to present (v1r2 and later) are listed/defined first. If a variable name/definition was different for v1r1, it is given in parentheses.

The primary variables of interest to most users include (in order of appearance in Table 3):

- QC_Flag (Byte1 for v1r1)
- PQI2 (Byte3 for v1r1)
- PQI4 (Byte5 for v1r1)
- SAAI (DAII for v1r1)
- Dust
- Latitude
- Longitude
- Smoke

Since the VIIRS ADP algorithm allows for two detection paths, users can work with VIIRS ADP data in two primary ways: (1) identify the presence only (yes/no) of dust and smoke aerosols via the IR-visible path or (2) identify the presence and relative intensity of dust and smoke aerosols via the deep-blue path. Figure 2 has examples of displaying VIIRS ADP data for these two options, and examples of IDL and Python code for processing VIIRS ADP data are given in the Appendix.

For option (1), use the following variables (**bolded**) in the VIIRS ADP data file:

- **Latitude**
- **Longitude**
- For smoke aerosols:
    - **Smoke** (see Table 3)
        - Select pixels = 1 for smoke present
- For dust aerosols:
    - **Dust** (see Table 3)
        - Select pixels = 1 for dust present
    - **PQI2** (**Byte3** for v1r1), bit 1 (see Table 6)
        - Select pixels = 0 for outside of sun glint region

For option (2), use the following variables (**bolded**) in the VIIRS ADP data file:

- **SAAI** (**DAII** for v1r1)
    - Display SAAI (DAII) in the range of [0,2] for smoke (thin to thick intensity)
    - Display SAAI (DAII) in the range of [0,5] for dust (thin to thick intensity)
- **Latitude**
- **Longitude**
- For smoke aerosols:
    - **Smoke** (see Table 3)
        - Select pixels = 1 for smoke present
    - **PQI4** (**Byte5** for v1r1), bits 4-5 (see Table 8)
        - Select pixels = 0 for deep-blue algorithm path **and** = 48 for both algorithm paths
- For dust aerosols:
    - **Dust** (see Table 3)
        - Select pixels = 1 for dust present
    - **PQI4** (**Byte5** for v1r1), bits 6-7 (see Table 8)
        - Select pixels = 0 for deep-blue algorithm path **and** = 192 for both algorithm paths
    - **PQI2** (**Byte3** for v1r1), bit 1 (see Table 6)
        - Select pixels = 0 for outside of sun glint region

The VIIRS ADP data file also includes a confidence flag, the QC_Flag (Byte1 for v1r1) variable, which allows users to select high, medium, or low quality pixels. For **qualitative** applications, all qualities (high + medium + low) may be used; this is the default setting, so QC_Flag (Byte1 for v1r1) does not need to be set explicitly for qualitative use. For **quantitative** applications, the Top 2 qualities (high + medium) are recommended, as follows:

- For smoke pixels:
    - **QC_Flag** (**Byte1** for v1r1), bits 2-3 (see Table 4)
        - Select pixels = 0 (= 12 for v1r1) for high quality **and** = 4 (= 8 for v1r1) for medium quality
- For dust pixels:
    - **QC_Flag** (**Byte1** for v1r1), bits 4-5 (see Table 4)
        - Select pixels = 0 (= 48 for v1r1) for high quality **and** = 16 (= 32 for v1r1) for medium quality

Table 3. Variables output from the VIIRS EPS ADP algorithm. Variable names in parentheses are valid for product version v1r1.

Variable	Type	Description	Dim	Units	Range
Ash	Byte	Volcanic Ash Flag: 1 = yes, 0 = No	2	1	0,1
AshConfidHighPct	Float	Percent of high confidence ash	0	Percent	0, 100
AshConfidLowPct	Float	Percent of low confidence ash	0	Percent	0, 100
AshConfidMediumPct	Float	Percent of medium confidence ash	0	Percent	0, 100
AshPct	Float	Percent of good ash retrieval	0	Percent	0, 100
QC_Flag (Byte1)	Byte	Quality Flag for Ash, Smoke, Dust and NUC (see Table 3)	2	1	-128,127
PQI1 (Byte2)	Byte	Product Quality Information (see Table 4)	2	1	-128,127
PQI2 (Byte3)	Byte	Product Quality Information (see Table 5)	2	1	-128,127
PQI3 (Byte4)	Byte	Product Quality Information (see Table 6)	2	1	-128,127
PQI4 (Byte5)	Byte	Product Quality Information (see Table 7)	2	1	-128,127
Cloud	Byte	Cloud Flag: 1 = yes, 0 = no	2	1	0,1
SAAI (DAII)	Float	Scaled Absorbing Aerosol Index	2	1	
Dust	Byte	Dust flag: 1 = yes, 0 = no	2	1	0,1
DustConfidHighPct	Float	Percent of high confidence dust	0	Percent	0, 100
DustConfidLowPct	Float	Percent of low confidence dust	0	Percent	0, 100
DustConfidMediumPct	Float	Percent of medium confidence dust	0	Percent	0, 100
DustPct	Float	Percent of good dust retrieval	0	Percent	0, 100
Latitude	Float	Pixel latitude in field latitude	2	° North	-90, 90
Longitude	Float	Pixel longitude in field longitude	2	° East	-180, 180
DSDI (NDAI)	Float	Dust Smoke Discrimination Index	2	1	
NUC	Byte	None, Unknown, Clear_sky Flag: 1 = yes, 0 = no	2	1	0, 1
NUCConfidHighPct	Float	Percent of high confidence NUC	0	Percent	0, 100
NUCConfidLowPct	Float	Percent of low confidence NUC	0	Percent	0, 100
NUCConfidMediumPct	Float	Percent of medium confidence NUC	0	Percent	0, 100
NUCPct	Float	Percent of good NUC retrieval	0	Percent	0, 100
NoAshPct	Float	Percent of ash not determined (bad)	0	Percent	0, 100
NoDustPct	Float	Percent of dust not determined (bad)	0	Percent	0, 100
NoNUCPct	Float	Percent of NUC not determined (bad)	0	Percent	0, 100
NoSmokePct	Float	Percent of smoke not determined (bad)	0	Percent	0, 100
NumOfGoodAshRetrieval	Long	Number of Good Ash Retrievals	0	1	
NumOfGoodDustRetrieval	Long	Number of Good Dust Retrievals	0	1	

Variable	Type	Description	Dim	Units	Range
NumOfGoodNUCRetrieval	Long	Number of Good NUC Retrievals	0	1	
NumOfGoodSmokeRetrieval	Long	Number of Good Smoke Retrievals	0	1	
NumOfQualityFlag	Long	Number of quality flags	0	1	
NumOfSatZenAngLess60	Long	Number of pixels with satellite zenith angle < 60 degree	0	1	
NumOfSolZenAngLess60	Long	Number of pixels with solar zenith angle < 60 degree	0	1	
Smoke	Byte	Smoke Flag: 1 = yes, 0 = no	2	1	0, 1
SmokeCon	Float	Smoke Concentration	2	$\mu g/m^3$	
SmokeConfidHighPct	Float	Percent of high confidence smoke	0	Percent	0, 100
SmokeConfidLowPct	Float	Percent of low confidence smoke	0	Percent	0, 100
SmokeConfidMediumPct	Float	Percent of medium confidence smoke	0	Percent	0, 100
SmokePct	Float	Percent of good smoke retrieval	0	Percent	0, 100
SnowIce	Byte	Snow Ice Flag: 1 = yes, 0 = no	2	1	0, 1
StartColumn	Long	Start column index	0		
StartRow	Long	Start row index	0		
TotalPixel	Long	Total number of pixels where retrievals are attempted	0	1	

Table 4. Definitions of bit-wise quality flags for the "QC_Flag" variable (valid for product versions v1r2 and later) and "Byte1" variable (valid for product version v1r1 only).

Bit*	Quality Flag Name	Meaning (2-bits)							
		01		10		00		11	
		QC_Flag	Byte1	QC_Flag	Byte1	QC_Flag	Byte1	QC_Flag	Byte1
0-1	QC_ASH_CONFIDENCE	Low quality	Medium quality	Medium quality	Low quality	High quality	Default	Bad/ missing	High quality
2-3	QC_SMOKE_CONFIDENCE								
4-5	QC_DUST_CONFIDENCE								
6-7	QC_NUC_CONFIDENCE								

*Start from the least significant bit

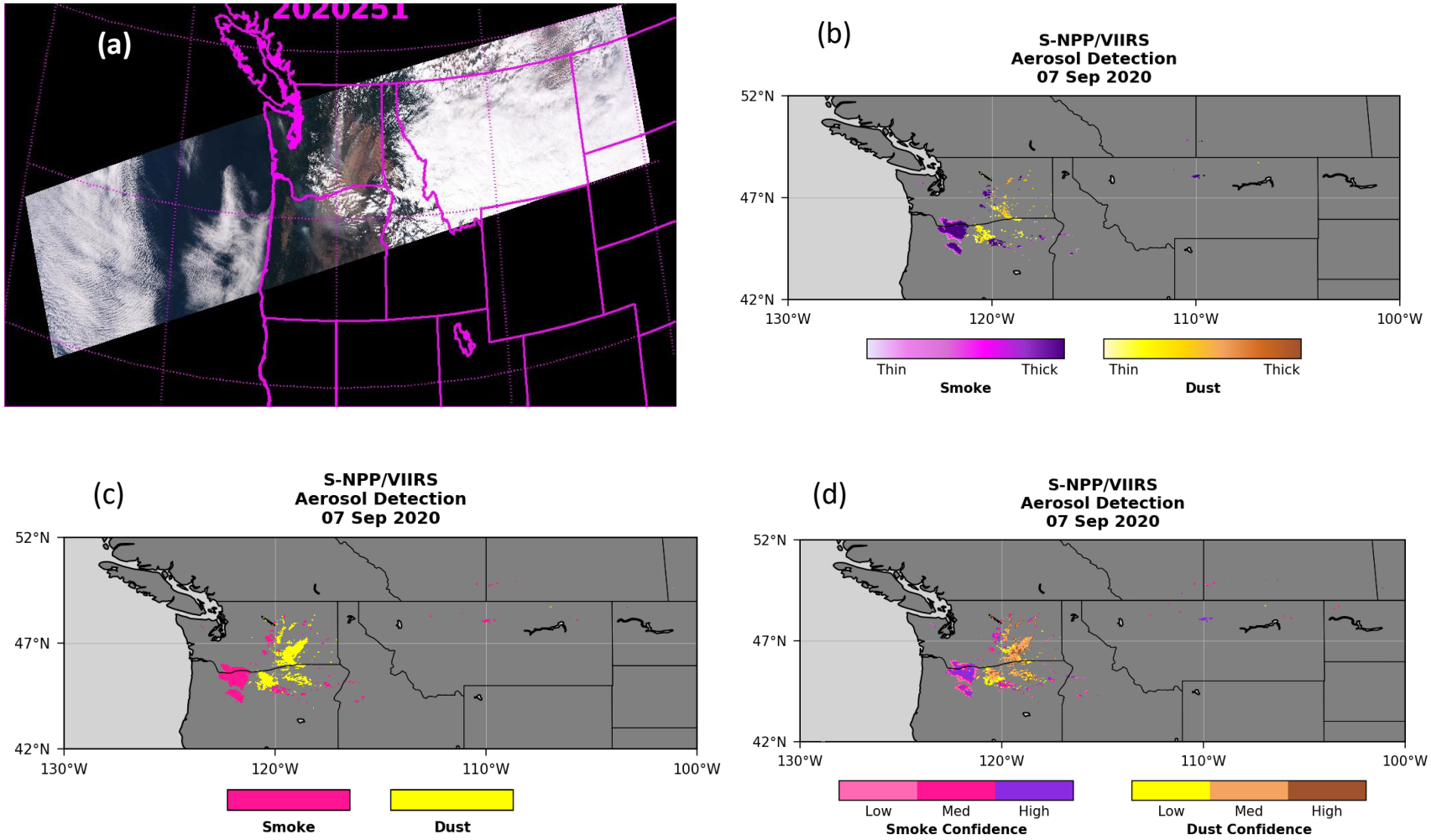

Figure 2. Examples of VIIRS APD display options for smoke and dust plumes in the western US on 7 September, 2020: (a) true color image showing visible smoke and blowing dust, (b) ADP relative smoke/dust intensity using the SAAI (DAII for v1r1) variable, (c) ADP presence of smoke/dust aerosols only, and (d) ADP high/medium/low confidence flags for presence of smoke/dust aerosols.

Table 5. Definitions of bit-wise diagnostic flags for the "PQI1" ("Byte2" for v1r1) variable.

Bit*	Diagnostic Flag Name	Meaning			
		1-bit	0 (default)	1	-----
		2-bits	00 (default)	01	11
0	QC_INPUT_LON	1-bit	Valid longitude	Invalid longitude (longitude > 180 or < -180)	-----
1	QC_INPUT_LAT	1-bit	Valid latitude	Invalid latitude (latitude > 90 or < -90)	-----
2-3	QC_INPUT_SOLZEN	2-bits	Valid solar zenith angle (SZA) (0 ≤ SZA ≤ 90)	Invalid SZA (SZA > 90 or < 0)	90 ≥ SZA > 60
4-5	QC_INPUT_SATZEN	2-bits	Valid local zenith angle (VZA) (0 ≤ VZA ≤ 90)	Invalid VZA (VZA > 90 or < 0)	90 ≥ VZA > 60
6-7	QC_INPUT_SNOW/ICE_SOURCE	2-bits	VIIRS snow/ice mask	IMS snow/ice mask	Internal snow/ice mask

*Start from the least significant bit

Table 6. Definitions of bit-wise diagnostic flags for the "PQI2" ("Byte3" for v1r1) variable.

Bit*	Diagnostic Flag Name	Meaning (1-bit)	
		0 (default)	1
0	QC_INPUT_SUNGLINT_SOURCE	VIIRS sun glint mask (from Cloud Mask product)	Internal sun glint mask
1	QC_INPUT_SUNGLINT	Outside of sun glint	Within sun glint
2	QC_INPUT_LAND/WATER	Water	Land
3	QC_INPUT_DAY/NIGHT	Day	Night
4	QC_WATER_SMOKE_INPUT	Valid VIIRS inputs	Invalid VIIRS inputs
5	QC_WATER_SMOKE_CLOUD	Cloud-free	Obscured by clouds
6	QC_WATER_SMOKE_SNOW/ICE	Snow/ice free	With snow/ice
7	QC_WATER_SMOKE_TYPE (only for IR/Visible algorithm path)	Thin Smoke	Thick Smoke

*Start from the least significant bit

Table 7. Definitions of bit-wise diagnostic flags for the "PQI3" ("Byte4" for v1r1) variable.

Bit*	Diagnostic Flag Name	Meaning (1-bit)	
		0 (default)	1
0	QC_WATER_DUST_INPUT	Valid VIIRS inputs	Invalid VIIRS inputs
1	QC_WATER_DUST_CLOUD	Cloud-free	Obscured by clouds
2	QC_WATER_DUST_SNOW/ICE	Snow/ice free	With snow/ice
3	QC_WATER_DUST_TYPE (only for IR/Visible algorithm path)	Thin dust	Thick dust
4	QC_LAND_SMOKE_INPUT	Invalid VIIRS inputs	Valid VIIRS inputs
5	QC_LAND_SMOKE_CLOUD	Cloud-free	Obscured by clouds
6	QC_LAND_SMOKE_SNOW/ICE	Snow/ice free	With snow/ice
7	QC_LAND_SMOKE_TYPE (only for IR/Visible algorithm path)	Fire	Thick smoke

*Start from the least significant bit

Table 8. Definitions of bit-wise diagnostic flags for the "PQI4" ("Byte5" for v1r1) variable.

Bit*	Diagnostic Flag Name	Meaning				
		1-bit	0 (default)	1	-----	-----
		2-bits	00 (default)	10	01	11
0	QC_LAND_DUST_INPUT	1-bit	Valid VIIRS inputs	Invalid VIIRS inputs	-----	-----
1	QC_LAND_DUST_CLOUD	1-bit	Cloud-free	Obscured by clouds	-----	-----
2	QC_LAND_DUST_SNOW/ICE	1-bit	Snow/ice free	With snow/ice	-----	-----
3	QC_LAND_DUST_TYPE (only for IR/Visible algorithm path)	1-bit	Thin dust	Thick dust	-----	-----
4-5	Smoke_Detection_Algorithm_Path	2-bits	Deep-blue	Missing	IR-Visible	Both
6-7	Dust_Detection_Algorithm_Path	2-bits				

*Start from the least significant bit

8. Data Availability

The primary source for the VIIRS aerosol products is NOAA's Comprehensive Large Array-data Stewardship System (CLASS) web interface (www.class.noaa.gov). ADP data are available in CLASS beginning July 6, 2017 for SNPP/VIIRS and beginning March 7, 2019 for NOAA-20/VIIRS. Starting at 00:00 UTC on August 13, 2018, several of the key SNPP/VIIRS ADP variable names and definitions changed, as described in Section 7. For the period January 31 to June 13, 2019, use the SNPP and NOAA-20 VIIRS smoke ADP with caution, as a bug was present in the algorithm that caused false smoke detections over the ocean.

Videos with instructions on how to order and download VIIRS data from CLASS are available from the AerosolWatch YouTube channel. The steps required to order VIIRS EPS ADP files (illustrated in Figures 3 and 4) are summarized here, for reference:

1. If you have not already done so, register to become a user of CLASS.
2. Login with your username and password.
3. Set "User Preferences": (1) check "No" on the "Package Geolocation with JPSS Data Products" option since the ADP file contains the longitude and latitude and (2) check "Yes" on the "De-aggregate JPSS Data Products" for the 86-second-granule-level files.
4. From the drop-down product search list on the home page, select "JPSS VIIRS Products (Granule)(JPSS_GRAN)" and click the ">>GO" button on the right.
5. Once on the "Search – JPSS_GRAN" page, select the spatial domain of interest by drawing a rectangle on the map, or by specifying the longitude and latitude domains on the right.
6. Select the time period by changing the "Start/End Date/Time".
7. Select the "VIIRS Aerosol Detection EDR" datatype from the "Advanced Search" list (you may need to expand the list by clicking the + button).
8. Select the satellite, either "S-NPP" or "NOAA-20".
9. Click "Quick Search & Order" or "Search" button to order the data.

The "Search" option is useful if you would like to view the inventory listing of the data available within your search parameters and select a small number of specific files. If you are confident in your search parameters, use the "Quick Search and Order" button, which will skip the inventory list.

Once the ordered data are ready, you will receive an email instructing you how to download the data via anonymous FTP. For larger order sizes, you can request Block Orders through the CLASS helpdesk (class.help@noaa.gov). Since these data requests must be manually pulled from the tape library, CLASS handles the bulk orders on a best effort basis. Finally, subscriptions are also available to users who require regular data access.

Note that beginning on April 11, 2018, VIIRS Enterprise ADP data are available in CLASS as TAR files, each of which contains several individual granules (individual NetCDF files) corresponding to roughly 10 minutes of VIIRS scans. This change was made to allow for easier access to the data, since CLASS order limits are restricted by file counts (500 maximum). Daily TAR files of the products for the most recent 90 days are also available via anonymous FTP download at ftp://ftp-npp.bou.class.noaa.gov/.

9. Known Issues to Date

The VIIRS Aerosol team has identified the following APD data quality problems that the users should be aware of:

A. False dust detections may appear for thick smoke plumes (seen as brown-colored plumes in RGB imagery) over both land and water.
B. Sulfur dioxide (SO_2) plumes from volcanic eruptions may be identified as smoke.
C. Thin dust plumes over vegetated surfaces may be mis-classified as smoke.

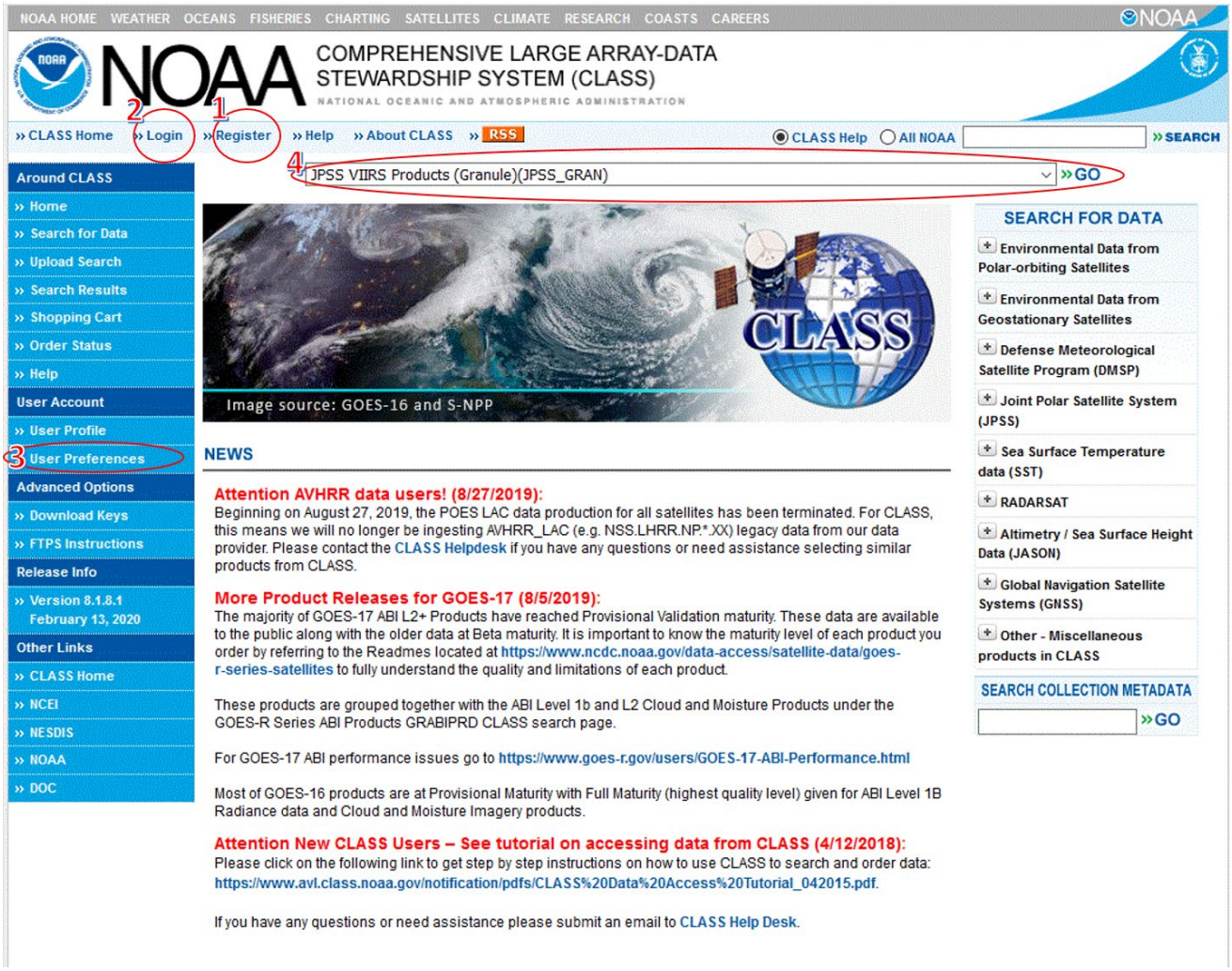

Figure 3. NOAA's CLASS homepage showing initial steps to order VIIRS data.

Figure 4. Ordering VIIRS Enterprise ADP data from NOAA's CLASS web page.

Appendix: Helpful Tools for Working with VIIRS ADP Files

A. NetCDF Tools

For users unaccustomed to working with NetCDF4 formatted files, please visit the website https://www.unidata.ucar.edu/software/netcdf/ for information.

B. Panoply Data Viewer

The Panoply NetCDF, HDF and GRIB Data Viewer developed by NASS GISS is a convenient tool for visualization of the EPS ADP outputs. Please visit the website https://www.giss.nasa.gov/tools/panoply/ for more information about this software.

C. IDL Tools

IDL has a built-in library of commands for NetCDF4 files. Documentation can be found online at https://www.harrisgeospatial.com/docs/NCDF_Overview.html or using IDL Help.

Also, Michael Galloy has written a particularly helpful IDL program to read HDF5 (also works for NetCDF4) arrays into IDL. It is available at http://docs.idldev.com/idllib/hdf5/mg_h5_getdata-code.html and provided below for convenience.

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Routine for extracting datasets, slices of datasets, or attributes from
; an HDF 5 file with simple notation.
;
; :Todo:
;    better error messages when items not found
;
; :Categories:
;    file i/o, hdf5, sdf
;
; :Examples:
;    An example file is provided with the IDL distribution::
;
;       IDL> f = filepath('hdf5_test.h5', subdir=['examples', 'data'])
;
;    A full dataset can be easily extracted::
;
;       IDL> fullResult = mg_h5_getdata(f, '/arrays/3D int array')
;
```

```
;    Slices can also be pulled out::
;
;       IDL> bounds = [[3, 3, 1], [5, 49, 2], [0, 49, 3]]
;       IDL> res1 = mg_h5_getdata(f, '/arrays/3D int array', bounds=bounds)
;       IDL> help, res1
;       RESULT1         LONG      = Array[1, 23, 17]
;
;    Verify that the slice is the same as the slice pulled out of the
;    fullResult::
;
;       IDL> same = array_equal(fullResult[3, 5:*:2, 0:49:3], res1)
;       IDL> print, same ? 'equal' : 'error'
;       equal
;
;    Normal IDL array indexing notation can be used as well::
;
;       IDL> bounds = '3, 5:*:2, 0:49:3'
;       IDL> res2 = mg_h5_getdata(f, '/arrays/3D int array', bounds=bounds)
;       IDL> print, array_equal(res1, res2) ? 'equal' : 'error'
;       equal
;
;    The variable location and bounds can be combined to slice a variable::
;
;       IDL> res3 = mg_h5_getdata(f, '/arrays/3D int array[3, 5:*:2, 0:49:3]')
;       IDL> print, array_equal(res1, res3) ? 'equal' : 'error'
;       equal
;
;    Attributes can be accessed as well::
;
;       IDL> print, mg_h5_getdata(f, '/images/Eskimo.CLASS')
;       IMAGE
;
;    This example is available as a main-level program included in this file::
;
;       IDL> .run mg_h5_getdata
;
; :Author:
;    Michael Galloy
;
; :Copyright:
```

```
;    This library is released under a BSD-type license.
;
;    Copyright (c) 2007-2010, Michael Galloy <mgalloy@idldev.com>
;
;    All rights reserved.
;
;    Redistribution and use in source and binary forms, with or without
;    modification, are permitted provided that the following conditions are
;    met:
;
;        a. Redistributions of source code must retain the above copyright
;           notice, this list of conditions and the following disclaimer.
;        b. Redistributions in binary form must reproduce the above copyright
;           notice, this list of conditions and the following disclaimer in
;           the documentation and/or other materials provided with the
;           distribution.
;        c. Neither the name of Michael Galloy nor the names of its
;           contributors may be used to endorse or promote products derived
;           from this software without specific prior written permission.
;
;    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
;    IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
;    THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
;    PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
;    CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
;    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
;    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
;    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
;    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
;    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
;    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
;-

;+
; Converts normal IDL indexing notation (represented as a string) into a
; `lonarr(ndims, 3)` where the first row is start values, the second row is
; the end values, and the last row is the stride value.
;
; :Private:
;
```

```
; :Returns:
;    lonarr(ndims, 3)
;
; :Params:
;    sbounds : in, required, type=string
;       bounds specified as a string using IDL's normal indexing notation
;
; :Keywords:
;    dimensions : in, optional, type=lonarr(ndims)
;       dimensions of the full array; required if a '*' is used in sbounds
;    single : out, optional, type=boolean
;       set to a named variable to determine if the bounds expression was
;       specified in single-index dimensioning
;===============================================================================
function mg_h5_getdata_convertbounds, sbounds, dimensions=dimensions, $
                                      single=single
  compile_opt strictarr
  on_error, 2

  dimIndices = strtrim(strsplit(sbounds, ',', /extract, count=ndims), 2)
  result = lonarr(ndims, 3)

  case ndims of
    1: begin
        single = 1B
        args = strsplit(dimIndices[0], ':', /extract, count=nargs)
        if (args[0] eq '*') then begin
          result[0, *] = [0, product(dimensions) - 1L, 1L]
        endif else begin
          result[0, *] = [long(args[0]), long(args[0]), 1L]
        endelse
      end
    n_elements(dimensions): begin
        single = 0B
        for d = 0L, ndims - 1L do begin
          args = strsplit(dimIndices[d], ':', /extract, count=nargs)
          case nargs of
            1 : begin
                if (args[0] eq '*') then begin
                  result[d, *] = [0, dimensions[d] - 1L, 1L]
```

```
              endif else begin
                result[d, *] = [long(args[0]), long(args[0]), 1L]
              endelse
            end
          2 : begin
              if (args[1] eq '*') then begin
                result[d, *] = [long(args[0]), dimensions[d] - 1L, 1L]
              endif else begin
                result[d, *] = [long(args[0]), long(args[1]), 1L]
              endelse
            end
          3 : begin
              if (args[1] eq '*') then begin
                result[d, *] = [long(args[0]), dimensions[d] - 1L, long(args[2])]
              endif else begin
                result[d, *] = long(args)
              endelse
            end
          else: message, 'invalid array indexing notation'
        endcase
      endfor
    end
  else:  message, 'invalid number of dimensions in array indexing notation'
endcase

return, result
end


;+
; Compute the H5D_SELECT_HYPERSLAB arguments from the bounds.
;
; :Private:
;
; :Params:
;    bounds : in, required, type="lonarr(ndims, 3)"
;       bounds
;
; :Keywords:
;    start : out, optional, type=lonarr(ndims)
```

```
;       input for start argument to H5S_SELECT_HYPERSLAB
;    count : out, optional, type=lonarr(ndims)
;       input for count argument to H5S_SELECT_HYPERSLAB
;    block : out, optional, type=lonarr(ndims)
;       input for block keyword to H5S_SELECT_HYPERSLAB
;    stride : out, optional, type=lonarr(ndims)
;       input for stride keyword to H5S_SELECT_HYPERSLAB
;-
pro mg_h5_getdata_computeslab, bounds, $
                               start=start, count=count, $
                               block=block, stride=stride
  compile_opt strictarr

  ndims = (size(bounds, /dimensions))[0]

  start = reform(bounds[*, 0])
  stride = reform(bounds[*, 2])

  count = ceil((bounds[*, 1] - bounds[*, 0] + 1L) / float(bounds[*, 2])) > 1
  block = lonarr(ndims) + 1L
end


;+
; Reads data in a dataset.
;
; :Private:
;
; :Returns:
;    value of data read from dataset
;
; :Params:
;    fileId : in, required, type=long
;       HDF 5 indentifier of the file
;    variable : in, required, type=string
;       string navigating the path to the dataset
;
; :Keywords:
;    bounds : in, optional, type="lonarr(3, ndims) or string"
;       gives start value, end value, and stride for each dimension of the
```

```
;       variable
;    error : out, optional, type=long
;       error value
;-
function mg_h5_getdata_getvariable, fileId, variable, bounds=bounds, $
                                    error=error
  compile_opt strictarr

  catch, error
  if (error ne 0L) then begin
    catch, /cancel
    error = 1L
    return, -1L
  endif

  variableId = h5d_open(fileId, variable)
  variableSpace = h5d_get_space(variableId)

  fullBounds = h5s_get_select_bounds(variableSpace)
  sz = size(fullBounds, /dimensions)
  fullBounds = [[fullBounds], [lonarr(sz[0]) + 1L]]
  dimensions = reform(fullBounds[*, 1] - fullBounds[*, 0] + 1L)

  case size(bounds, /type) of
    0 : _bounds = fullBounds
    7 : _bounds = mg_h5_getdata_convertbounds(bounds, dimensions=dimensions, single=single)
    else: _bounds = transpose(bounds)
  endcase

  if (keyword_set(single)) then begin
    ; TODO: implement
    message, 'single-dimension indices not implemented yet'
  endif else begin
    mg_h5_getdata_computeslab, _bounds, $
                               start=start, count=count, $
                               block=block, stride=stride
    resultSpace = h5s_create_simple(count)

    h5s_select_hyperslab, variableSpace, start, count, $
                          block=block, stride=stride, /reset
```

```
  endelse

  data = h5d_read(variableId, $
                  file_space=variableSpace, $
                  memory_space=resultSpace)

  h5s_close, resultSpace
  h5s_close, variableSpace
  h5d_close, variableId

  return, data
end


;+
; Get the value of the attribute from its group, dataset, or type.
;
; :Private:
;
; :Returns:
;    attribute data
;
; :Params:
;    loc : in, required, type=long
;       identifier of group, dataset, or type that contains the attribute
;    attname : in, required, type=string
;       attribute name
;-
function mg_h5_getdata_getattributedata, loc, attname
  compile_opt strictarr
  on_error, 2

  att = h5a_open_name(loc, attname)
  result = h5a_read(att)
  h5a_close, att

  return, result
end
```

```
;+
; Get the value of an attribute in a file.
;
; :Private:
;
; :Returns:
;    attribute value
;
; :Params:
;    fileId : in, required, type=long
;       HDF 5 file identifier of the file to read
;    variable : in, required, type=string
;       path to attribute using "/" to navigate groups/datasets and "." to
;       indicate the attribute name
;
; :Keywords:
;    error : out, optional, type=long
;       error value
;-
function mg_h5_getdata_getattribute, fileId, variable, error=error
  compile_opt strictarr

  catch, error
  if (error ne 0L) then begin
    catch, /cancel
    error = 1L
    return, -1L
  endif

  tokens = strsplit(variable, '.', escape='\', count=ndots)
  dotpos = tokens[1L] - 1L

  loc = strmid(variable, 0, dotpos)
  attname = strmid(variable, dotpos + 1L)
  objInfo = h5g_get_objinfo(fileId, loc)

  result = -1L

  case objInfo.type of
    'LINK': message, 'Cannot handle an attribute of a reference'
```

```
    'GROUP': begin
        group = h5g_open(fileId, loc)
        result = mg_h5_getdata_getattributedata(group, attname)
        h5g_close, group
      end
    'DATASET': begin
        dataset = h5d_open(fileId, loc)
        result = mg_h5_getdata_getattributedata(dataset, attname)
        h5d_close, dataset
      end
    'TYPE': begin
        type = h5t_open(fileId, loc)
        result = mg_h5_getdata_getattributedata(type, attname)
        h5t_close, type
      end
    'UNKNOWN': message, 'Unknown item'
  endcase

  return, result
end

;============================================================================

;+
; Pulls out a section of a HDF5 variable.
;
; :Returns:
;    data array
;
; :Params:
;    filename : in, required, type=string
;       filename of the HDF5 file
;    variable : in, required, type=string
;       variable name (with path if inside a group)
;
; :Keywords:
;    bounds : in, optional, type="lonarr(3, ndims) or string"
;       gives start value, end value, and stride for each dimension of the
;       variable
;    error : out, optional, type=long
```

```
;       error value
;-
function mg_h5_getdata, filename, variable, bounds=bounds, error=error
  compile_opt strictarr
  on_error, 2

  IF ~File_Test(filename) THEN Return, -1
  fileId = h5f_open(filename)

  tokens = strsplit(variable, '.', escape='\', count=ndots)

  if (ndots eq 1L) then begin
    ; variable
    bracketPos = strpos(variable, '[')
    if (bracketPos eq -1L) then begin
      _variable = variable
      if (n_elements(bounds) gt 0L) then _bounds = bounds
    endif else begin
      _variable = strmid(variable, 0L, bracketPos)
      closedBracketPos = strpos(variable, ']', /reverse_search)
      _bounds = strmid(variable, bracketPos + 1L, closedBracketPos - bracketPos - 1L)
    endelse

    result = mg_h5_getdata_getvariable(fileId, _variable, bounds=_bounds, $
                                       error=error)
    if (error) then message, 'variable not found', /informational
  endif else begin
    ; attribute
    result = mg_h5_getdata_getattribute(fileId, variable, error=error)
    if (error) then message, 'attribute not found', /informational
  endelse

  h5f_close, fileId

  return, result
end
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

D. Example of IDL Code for Processing VIIRS ADP (written by Pubu Ciren)

```
;=============================================================================================================
;main code to read ADP data
||
;=============================================================================================================
pro read_JPSS_EPS_ADP
   MISVAL=-999.9

   filename='JRR-ADP_v2r0_j01_s201804081551293_e201804081552538_c201808151527460.nc'
; get version no from the file
Version=strmid(filename,7,4)

         lat = mg_h5_getdata(filename,'Latitude')
         lon = mg_h5_getdata(filename,'Longitude')
         Ash = mg_h5_getdata(filename,'Ash')
;smoke/dust mask (1/0 - YES/NO)
         smoke=mg_h5_getdata(filename,'Smoke')
          dust=mg_h5_getdata(filename,'Dust')
         cld = mg_h5_getdata(filename,'Cloud')
         Nuc = mg_h5_getdata(filename,'NUC')
         snowice=mg_h5_getdata(filename,'SnowIce')
  if (Version eq 'v1r0' or  Version eq 'v1r1') then begin
;Before and on version 'v1r1'
         AAI = mg_h5_getdata(filename,'DAII')
         DSDI =mg_h5_getdata(filename,'NDAI')
         Qual_B1=mg_h5_getdata(filename,'Byte1')
         Qual_B2=mg_h5_getdata(filename,'Byte2')
         Qual_B3=mg_h5_getdata(filename,'Byte3')
         Qual_B4=mg_h5_getdata(filename,'Byte4')
         Qual_B5=mg_h5_getdata(filename,'Byte5')
 endif else begin
;After and on version 'v1r2'
         AAI = mg_h5_getdata(filename,'SAAI')
         DSDI =mg_h5_getdata(filename,'DSDI')
         Qual_B1=mg_h5_getdata(filename,'QC_Flag')
         Qual_B2=mg_h5_getdata(filename,'PQI1')
         Qual_B3=mg_h5_getdata(filename,'PQI2')
         Qual_B4=mg_h5_getdata(filename,'PQI3')
         Qual_B5=mg_h5_getdata(filename,'PQI4')
```

```
 endelse

;=============================================================================================================
;Ancillary information: sunglint mask(sungln,  0: out of glint; 1: inside glint), land/water mask (0: water, 1:
land, lndwat)
;=============================================================================================================
  sugln=smoke
  sugln(*,*)=0
  lndwat=smoke
  lndwat(*,*)=0
  lndwat=smoke
  lndwat(*,*)=0

; Byte3 bit 2 (sun glint)
   tmp_mask=Qual_B3
   idx = WHERE((tmp_mask AND 2) EQ 2, COMPLEMENT=cidx, nc)
   IF nc GT 0 THEN sugln[idx] = 1

; Byte3 bit 3 (land/water)
   tmp_mask=Qual_B3
   idx = WHERE((tmp_mask AND 4) EQ 4, COMPLEMENT=cidx, nc)
   IF nc GT 0 THEN lndwat[idx] = 1

;no glint on land
   idx=where(lndwat eq 1 and sugln eq 1,nn)
   if ( nn gt 0) then sugln(idx)=0

;=============================================================================================
; ADP INFORMATION; Three different ways to extract ADP information
;=============================================================================================
;1). smoke, dust type (yes/no flag: 1/0): Smoke_Type, Dust_Type
;=============================================================================================
    Smoke_Type=smoke
     Dust_Type=dust

; note that dust over sunglint area has to be removed in order to reduce false detection, user has to mask the
dust out with sunglint mask as following:
   idx=where(Dust_Type eq 1 and sugln eq 1, nn)
   if (nn gt 0) then begin
```

```
      Dust_Type(idx)=0
   endif

; User can also choose results from which the algorithm path to use here

;smoke/dust detection algorithm path (indicating the detected smoke/dust is from which algorithm path)

;Smoke_Type
;byte5, bits 4-5 (smoke detection algorithm source): 00: deep-blue only; 01:ir_visible only; 11: both
  tmp_mask=Qual_B5

   ;a. smoke detected by either deep-blue or IR_Visible Path (default)

   ;b. Choose smoke detected with deep-blue algorithm path
        idx=where(((ishft(tmp_mask,-4) and 3) eq 2),COMPLEMENT=cidx,n)
        if ( n gt 0) then smoke_Type(idx)=0

   ;c. Choose smoke detected with IR_Visible algorithm path
        idx=where(((ishft(tmp_mask,-4) and 3) eq 0),COMPLEMENT=cidx,n)
        if ( n gt 0) then smoke_Type(idx)=0

;dust_type
;byte5, bits 6-7 (Dust detection algorithm source): 00: deep-blue only; 01:ir_visible only; 11: both

  tmp_mask=Qual_B5

     ;a. Dust detected by either deep-blue or IR_Visible Path (default)

     ;b. Choose dust detected with deep-blue algorithm path

        idx=where(((ishft(tmp_mask,-6) and 3) eq 2), n)
        if (n gt 0 ) then Dust_Type(idx)=0

     ;c. Choose Dust detected with IR_Visible algorithm path
        idx=where(((ishft(tmp_mask,-6) and 3) eq 0),COMPLEMENT=cidx,n)
        if ( n gt 0 ) then Dust_Type(idx)=0

;=============================================================================================================
;2). smoke, dust type Quality (Confidence) level: Smoke_Qual  and  Dust_Qual;   1: Low; 2: Medium; and 3: High;
;=============================================================================================================
```

```
  Smoke_Qual=smoke
  Smoke_Qual(*,*)=0
  Dust_Qual=smoke
  Dust_Qual(*,*)=0

  if (Version eq 'v1r0' or  Version eq 'v1r1') then begin
;=================Before and on version 'v1r1'=====================================

; byte1 bits 3-4 (smoke quality: High)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-2) AND 3) EQ 3) and (Smoke_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Smoke_Qual[idx] = 3

;byte1, bits 5-6 (dust quality: High)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-4) AND 3) EQ 3) and (Dust_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Dust_Qual[idx] = 3


; byte1 bits 3-4 (smoke quality: Medium)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-2) AND 3) EQ 2) and (Smoke_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Smoke_Qual[idx] = 2


;byte1, bits 5-6 (dust quality: Medium)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-4) AND 3) EQ 2) and (Dust_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Dust_Qual[idx] = 2


; byte1 bits 3-4 (smoke quality: Low)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-2) AND 3) EQ 1) and (Smoke_type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Smoke_Qual[idx] = 1
```

```
;byte1, bits 5-6 (dust quality)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-4) AND 3) EQ 1) and (Dust_type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Dust_Qual[idx] = 1

  endif else begin
;=================After and on version 'v1r2'====================================
    ;smoke quality (High)
  idx=where(Smoke_Type eq 1, n)
   if (n gt 0) then Smoke_Qual(idx)=3
    ;dust quality  (High)
  idx=where(Dust_Type eq 1, n)
   if (n gt 0) then Dust_Qual(idx)=3


; byte1 bits 3-4 (smoke quality: Medium)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-2) AND 3) EQ 1) and (Smoke_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Smoke_Qual[idx] = 2

;byte1, bits 5-6 (dust quality: Medium)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-4) AND 3) EQ 1) and (Dust_Type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Dust_Qual[idx] = 2


; byte1 bits 3-4 (smoke quality: Low)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-2) AND 3) EQ 2) and (Smoke_type eq 1), COMPLEMENT=cidx, nc)
;print, nc
   IF nc GT 0 THEN Smoke_Qual[idx] = 1

;byte1, bits 5-6 (dust quality)
tmp_mask=Qual_B1
   idx = WHERE(((ishft(tmp_mask,-4) AND 3) EQ 2) and (Dust_type eq 1), COMPLEMENT=cidx, nc)
;print, nc
```

```
   IF nc GT 0 THEN Dust_Qual[idx] = 1

 endelse
;=============================================================

;=============================================================================================================
;3). smoke, dust type with AAI value in order to visualize smoke/dust intensity: Smoke_AAI and Dust_AAI
;=============================================================================================================

;;;;smoke/dust detection algorithm path (indicating the deteced smoke/dust is from which algorithm path),
;;;; AAI values is only available for pixles detected with deep-blue algorithm path
;smoke
;byte5, bits 4-5 (smoke detection algorithm source): 00: deep-blue 01:ir_visible 11: both

        ;a. Choose smoke detected with deep-blue algorithm path
            tmp_mask=Qual_B5
            idx=where(((ishft(tmp_mask,-4) and 3) eq 2),COMPLEMENT=cidx,n)
            if ( n gt 0) then Smoke(idx)=0
;dust
;byte5, bits 6-7 (smoke detection algorithm source): 00: deep-blue 01:ir_visible 11: both
        ;a. Choose dust detected with deep-blue algorithm path
            tmp_mask=Qual_B5
           idx=where(((ishft(tmp_mask,-6) and 3) eq 2), n)
           if (n gt 0) then dust(idx)=0


;dust (remove dust detection in the sunglint region)
        idx=where(dust eq 1 and sugln eq 1, nn)
        if (nn gt 0) then begin
           dust(idx)=0
        endif


; scaled Absorbing Aerosol Index for dust - show dust intensity
       Dust_AAI=AAI
       Dust_AAI(*,*)=MISVAL
       idx=where(dust eq 1,n)
       if (n gt 0) then Dust_AAI(idx)=AAI(idx)

; scaled Absorbing Aerosol Index for smoke - show smoke intensity
```

```
; I believe this is what you want to display if you want to display smoke/mask
; User can color-scaled this value from 0 to 2 for smoke and 0 to 5 for dust for best color-stretch.
       Smoke_AAI=AAI
       Smoke_AAI(*,*)=MISVAL
       idx=where(smoke eq 1,n)
   if (n gt 0) then Smoke_AAI(idx)=AAI(idx)


end
```

E. Example of Python Code for Processing VIIRS ADP (written by Amy Huff and Ryan Theurer)

```
#File settings
file_path = 'D://Data/2020/2020-07/20200726/VIIRS ADP/'
file_name = 'JRR ADP_v2r1_j01_s202007262332292_e202007262333537_c202007262352470.nc'

#Open single data file and set file name
fname = file_path + file_name
file_id = Dataset(fname)


##Function to select and process VIIRS ADP data, aerosol detection only (smoke and dust)
def VIIRS_ADP_Detect(file_id):
    #Select smoke pixels using the "Smoke" variable (mask pixels with smoke absent)
    #smoke present = 1, smoke absent = 0
    Smoke_Flag = file_id.variables['Smoke'][:,:]
    Smoke_Mask = (Smoke_Flag == 0)
    Smoke = np.ma.masked_where(Smoke_Mask, Smoke_Flag)

    #Select dust pixels using the "Dust" variable (mask pixels with dust absent)
    #dust present = 1, dust absent = 0
    Dust_Flag = file_id.variables['Dust'][:,:]
    Dust_Mask = (Dust_Flag == 0)
    Dust_Intermediate = np.ma.masked_where(Dust_Mask, Dust_Flag)

    #Remove dust pixels in sun-glint regions using the "PQI2" variable, bit 1 (mask pixels within sun-glint)
```

```
    #outside sun-glint = 0, within sun-glint = 2
    PQI2 = file_id.variables['PQI2'][:,:]
    Glint_Mask = (PQI2 & 2 == 2)
    Dust = np.ma.masked_where(Glint_Mask, Dust_Intermediate)

    #Read in latitude and longitude
    Lon = file_id.variables['Longitude'][:,:]
    Lat = file_id.variables['Latitude'][:,:]

    return Smoke, Dust, Lat, Lon


##Function to select and process VIIRS ADP data, smoke/dust intensity using scaled absorbing aerosol index (SAAI)
def VIIRS_ADP_SAAI(file_id):
    #Read in the SAAI data
    SAAI = file_id.variables['SAAI'][:,:]

    #Select smoke pixels using the "Smoke" variable (mask pixels with smoke absent)
    #smoke present = 1, smoke absent = 0
    Smoke_Flag = file_id.variables['Smoke'][:,:]
    Smoke_Mask = (Smoke_Flag == 0)
    SAAI_Smoke = np.ma.masked_where(Smoke_Mask, SAAI)

    #Select deep-blue based algorithm smoke pixels using the "PQI4" variable, bits 4-5 (mask missing and IR-vis
    based algorithm pixels)
    #missing = 16, IR-vis based algorithm = 32
    PQI4 = file_id.variables['PQI4'][:,:]
    Smoke_Algorithm_Mask = (((PQI4 & 16) + (PQI4 & 32) == 16) | ((PQI4 & 16) + (PQI4 & 32) == 32))
    SAAI_Smoke_Total = np.ma.masked_where(Smoke_Algorithm_Mask, SAAI_Smoke)

    #Select dust pixels using the "Dust" variable (mask pixels with dust absent)
    #dust present = 1, dust absent = 0
    Dust_Flag = file_id.variables['Dust'][:,:]
    Dust_Mask = (Dust_Flag == 0)
    SAAI_Dust = np.ma.masked_where(Dust_Mask, SAAI)

    #Select deep-blue based algorithm dust pixels using the "PQI4" variable, bits 6-7 (mask missing and IR-vis
    based algorithm pixels)
    #missing = 64, IR-vis based algorithm = 128
    Dust_Algorithm_Mask = (((PQI4 & 64) + (PQI4 & 128) == 64) | ((PQI4 & 64) + (PQI4 & 128) == 128))
```

```
SAAI_Dust_Intermediate = np.ma.masked_where(Dust_Algorithm_Mask, SAAI_Dust)

#Remove deep-blue algorithm dust pixels in sun-glint regions using the "PQI2" variable, bit 1 (mask pixels
within sun-glint)
#outside sun-glint = 0, within sun-glint = 2
PQI2 = file_id.variables['PQI2'][:,:]
Glint_Mask = (PQI2 & 2 == 2)
SAAI_Dust_Total = np.ma.masked_where(Glint_Mask, SAAI_Dust_Intermediate)

#Read in latitude and longitude
Lon = file_id.variables['Longitude'][:,:]
Lat = file_id.variables['Latitude'][:,:]

return SAAI_Smoke_Total, SAAI_Dust_Total, Lat, Lon
```