python-awips Documentation

Unidata

Dec 11, 2018

Contents

The python-awips package provides a data access framework for requesting grid and geometry datasets from an EDEX server.

AWIPS is a weather display and analysis package developed by the National Weather Service for operational forecasting. UCAR's Unidata Program Center supports a non-operational open-source release of the AWIPS software (EDEX, CAVE, and python-awips).

CHAPTER 1

Pip Install

- pip install python-awips

CHAPTER 2

Conda Environment Install

To install the latest version of python-awips, with all required and optional packages:

- git clone https://github.com/Unidata/python-awips.git
- cd python-awips
- conda env create -f environment.yml
- source activate python-awips
- python setup.py install –force
- jupyter notebook examples

CHAPTER 3

Requirements

- python 2.7+
- numpy
- shapely
- six

3.1 Quick Example

```
from awips.dataaccess import DataAccessLayer
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
dataTypes = DataAccessLayer.getSupportedDatatypes()
list(dataTypes)

['acars',
 'binlightning',
 'bufrmosavn',
 'bufrmoseta',
 'bufrmosgfs',
 'bufrmoshpc',
 'bufrmoslamp',
 'bufrmosmrf',
 'bufrua',
 'climate',
 'common_obs_spatial',
 'gfe',
 'gfeeditarea',
 'grid',
 'maps',
 'modelsounding',
 'obs',
 'practicewarning',
```

(continues on next page)

(continued from previous page)

```
 'profiler',
 'radar',
 'radar_spatial',
 'satellite',
 'sfcobs',
 'topo',
 'warning']


request = DataAccessLayer.newDataRequest()
request.setDatatype("satellite")
availableSectors = DataAccessLayer.getAvailableLocationNames(request)
availableSectors.sort()
for sector in availableSectors:
    print sector
    request.setLocationNames(sector)
    availableProducts = DataAccessLayer.getAvailableParameters(request)
    availableProducts.sort()
    for product in availableProducts:
        print " - " + product

ECONUS
 - ACTP
 - ADP
 - AOD
 - CAPE
 - CH-01-0.47um
 - CH-02-0.64um
 - CH-03-0.87um
 - CH-04-1.38um
 ...
EFD
 - ACTP
 - ADP
 - AOD
 - CAPE
 - CH-01-0.47um
 - CH-02-0.64um
 - CH-03-0.87um
 - CH-04-1.38um
 ...
```

See the API Documentation for more information.

3.1.1 Read The Docs Contents

API Documentation

DataAccessLayer

awips.dataaccess.DataAccessLayer.**changeEDEXHost**(*newHostName*)

Changes the EDEX host the Data Access Framework is communicating with. Only works if using the native Python client implementation, otherwise, this method will throw a TypeError.

Args: newHostName: the EDEX host to connect to

awips.dataaccess.DataAccessLayer.**getAvailableLevels**(*request*)
Gets the available levels that match the request without actually requesting the data.

Args: request: the request to find matching levels for

Returns: a list of strings of available levels.

awips.dataaccess.DataAccessLayer.**getAvailableLocationNames**(*request*)
Gets the available location names that match the request without actually requesting the data.

Args: request: the request to find matching location names for

Returns: a list of strings of available location names.

awips.dataaccess.DataAccessLayer.**getAvailableParameters**(*request*)
Gets the available parameters names that match the request without actually requesting the data.

Args: request: the request to find matching parameter names for

Returns: a list of strings of available parameter names.

awips.dataaccess.DataAccessLayer.**getAvailableTimes**(*request*, *refTimeOnly=False*)
Get the times of available data to the request.

Args: request: the IDataRequest to get data for refTimeOnly: optional, use True if only unique refTimes should be

returned (without a forecastHr)

Returns: a list of DataTimes

awips.dataaccess.DataAccessLayer.**getForecastRun**(*cycle*, *times*)
Get the latest forecast run (list of objects) from all all cycles and times returned from DataAccessLayer "grid" response.

Args: cycle: Forecast cycle reference time times: All available times/cycles

Returns: DataTime array for a single forecast run

awips.dataaccess.DataAccessLayer.**getGeometryData**(*request*, *times=[]*)
Gets the geometry data that matches the request at the specified times. Each combination of geometry, level, and dataTime will be returned as a separate IGeometryData.

Args: request: the IDataRequest to get data for times: a list of DataTimes, a TimeRange, or None if the data is time

agnostic

Returns: a list of IGeometryData

awips.dataaccess.DataAccessLayer.**getGridData**(*request*, *times=[]*)
Gets the grid data that matches the request at the specified times. Each combination of parameter, level, and dataTime will be returned as a separate IGridData.

Args: request: the IDataRequest to get data for times: a list of DataTimes, a TimeRange, or None if the data is time

agnostic

Returns: a list of IGridData

awips.dataaccess.DataAccessLayer.**getIdentifierValues**(*request*, *identifierKey*)
Gets the allowed values for a particular identifier on this datatype.

Args: request: the request to find identifier values for identifierKey: the identifier to find values for

Returns: a list of strings of allowed values for the specified identifier

awips.dataaccess.DataAccessLayer.**getMetarObs**(*response*)
Processes a DataAccessLayer "obs" response into a dictionary, with special consideration for multi-value parameters "presWeather", "skyCover", and "skyLayerBase".

Args: response: DAL getGeometry() list

Returns: A dictionary of METAR obs

awips.dataaccess.DataAccessLayer.**getOptionalIdentifiers**(*request*)
Gets the optional identifiers for this request.

Args: request: the request to find optional identifiers for

Returns: a list of strings of optional identifiers

awips.dataaccess.DataAccessLayer.**getRadarProductIDs**(*availableParms*)
Get only the numeric idetifiers for NEXRAD3 products.

Args: availableParms: Full list of radar parameters

Returns: List of filtered parameters

awips.dataaccess.DataAccessLayer.**getRadarProductNames**(*availableParms*)

Get only the named idetifiers for NEXRAD3 products.

Args: availableParms: Full list of radar parameters

Returns: List of filtered parameters

awips.dataaccess.DataAccessLayer.**getRequiredIdentifiers**(*request*)
Gets the required identifiers for this request. These identifiers must be set on a request for the request of this datatype to succeed.

Args: request: the request to find required identifiers for

Returns: a list of strings of required identifiers

awips.dataaccess.DataAccessLayer.**getSupportedDatatypes**()
Gets the datatypes that are supported by the framework

Returns: a list of strings of supported datatypes

awips.dataaccess.DataAccessLayer.**getSynopticObs**(*response*)
Processes a DataAccessLayer "sfcobs" response into a dictionary of available parameters.

Args: response: DAL getGeometry() list

Returns: A dictionary of synop obs

awips.dataaccess.DataAccessLayer.**newDataRequest**(*datatype=None*, ***kwargs*)
Creates a new instance of IDataRequest suitable for the runtime environment. All args are optional and exist solely for convenience.

Args: datatype: the datatype to create a request for parameters: a list of parameters to set on the request levels: a list of levels to set on the request locationNames: a list of locationNames to set on the request envelope: an envelope to limit the request kwargs: any leftover kwargs will be set as identifiers

Returns: a new IDataRequest

awips.dataaccess.DataAccessLayer.**setLazyLoadGridLatLon**(*lazyLoadGridLatLon*)
Provide a hint to the Data Access Framework indicating whether to load the lat/lon data for a grid immediately or wait until it is needed. This is provided as a performance tuning hint and should not affect the way the Data Access Framework is used. Depending on the internal implementation of the Data Access Framework this hint might be ignored. Examples of when this should be set to True are when the lat/lon information is not used

or when it is used only if certain conditions within the data are met. It could be set to False if it is guaranteed that all lat/lon information is needed and it would be better to get any performance overhead for generating the lat/lon data out of the way during the initial request.

Args: lazyLoadGridLatLon: Boolean value indicating whether to lazy load.

IDataRequest (newDataRequest())

class awips.dataaccess.**IDataRequest**

An IDataRequest to be submitted to the DataAccessLayer to retrieve data.

__weakref__
list of weak references to the object (if defined)

addIdentifier(*key*, *value*)
Adds an identifier to the request. Identifiers are specific to the datatype being requested.

Args: key: the string key of the identifier value: the value of the identifier

getDatatype()
Gets the datatype of the request

Returns: the datatype set on the request

getEnvelope()
Gets the envelope on the request

Returns: a rectangular shapely geometry

getIdentifiers()
Gets the identifiers on the request

Returns: a dictionary of the identifiers

getLevels()
Gets the levels on the request

Returns: a list of strings of the levels

getLocationNames()
Gets the location names on the request

Returns: a list of strings of the location names

setDatatype(*datatype*)
Sets the datatype of the request.

Args: datatype: A string of the datatype, such as “grid”, “radar”, “gfe”, “obs”

setEnvelope(*env*)
Sets the envelope of the request. If supported by the datatype factory, the data returned for the request will be constrained to only the data within the envelope.

Args: env: a shapely geometry

setLevels(*levels*)
Sets the levels of data to request. Not all datatypes support levels.

Args: levels: a list of strings of level abbreviations to request

setLocationNames(*locationNames*)
Sets the location names of the request.

Args: locationNames: a list of strings of location names to request

setParameters(*params*)
Sets the parameters of data to request.

Args: params: a list of strings of parameters to request

PyData

class awips.dataaccess.PyData.**PyData**(*dataRecord*)

getAttribute(*key*)
Gets an attribute of the data.

Args: key: the key of the attribute

Returns: the value of the attribute

getAttributes()
Gets the valid attributes for the data.

Returns: a list of strings of the attribute names

getDataTime()
Gets the data time of the data.

Returns: the data time of the data, or None if no time is associated

getLevel()
Gets the level of the data.

Returns: the level of the data, or None if no level is associated

getLocationName()
Gets the location name of the data.

Returns: the location name of the data, or None if no location name is associated

PyGridData

class awips.dataaccess.PyGridData.**PyGridData**(*gridDataRecord*, *nx*, *ny*, *latLonGrid=None*, *latLonDelegate=None*)

getLatLonCoords()
Gets the lat/lon coordinates of the grid data.

Returns: a tuple where the first element is a numpy array of lons, and the second element is a numpy array of lats

getParameter()
Gets the parameter of the data.

Returns: the parameter of the data

getRawData(*unit=None*)
Gets the grid data as a numpy array.

Returns: a numpy array of the data

getUnit()
Gets the unit of the data.

Returns: the string abbreviation of the unit, or None if no unit is associated

PyGeometryData

class awips.dataaccess.PyGeometryData.**PyGeometryData**(*geoDataRecord*, *geometry*)

getGeometry()
Gets the geometry of the data.

Returns: a shapely geometry

getNumber(*param*)
Gets the number value of the specified param.

Args: param: the string name of the param

Returns: the number value of the param

getParameters()
Gets the parameters of the data.

Returns: a list of strings of the parameter names

getString(*param*)
Gets the string value of the specified param.

Args: param: the string name of the param

Returns: the string value of the param

getType(*param*)
Gets the type of the param.

Args: param: the string name of the param

Returns: a string of the type of the parameter, such as "STRING", "INT", "LONG", "FLOAT", or "DOUBLE"

getUnit(*param*)
Gets the unit of the specified param.

Args: param: the string name of the param

Returns: the string abbreviation of the unit of the param

ModelSounding

awips.dataaccess.ModelSounding.**changeEDEXHost**(*host*)
Changes the EDEX host the Data Access Framework is communicating with.

Args: host: the EDEX host to connect to

awips.dataaccess.ModelSounding.**getSounding**(*modelName*, *weatherElements*, *levels*, *samplePoint*, *timeRange=None*)
Performs a series of Data Access Framework requests to retrieve a sounding object based on the specified request parameters.

Args: modelName: the grid model datasetid to use as the basis of the sounding. weatherElements: a list of parameters to return in the sounding. levels: a list of levels to sample the given weather elements at samplePoint: a lat/lon pair to perform the sampling of data at. timeRange: (optional) a list of times, or a TimeRange to specify which forecast hours to use. If not specified, will default to all forecast hours.

Returns: A _SoundingCube instance, which acts a 3-tiered dictionary, keyed by DataTime, then by level and finally by weather element. If no data is available for the given request parameters, None is returned.

ThriftClientRouter

class awips.dataaccess.ThriftClientRouter.**LazyGridLatLon**(*client*, *nx*, *ny*, *envelope*, *crsWkt*)

class awips.dataaccess.ThriftClientRouter.**ThriftClientRouter**(*host='localhost'*)

getAvailableLevels(*request*)

getAvailableLocationNames(*request*)

getAvailableParameters(*request*)

getAvailableTimes(*request*, *refTimeOnly*)

getGeometryData(*request*, *times*)

getGridData(*request*, *times*)

getIdentifierValues(*request*, *identifierKey*)

getNotificationFilter(*request*)

getOptionalIdentifiers(*request*)

getRequiredIdentifiers(*request*)

getSupportedDatatypes()

newDataRequest(*datatype*, *parameters=[]*, *levels=[]*, *locationNames=[]*, *envelope=None*, ***kwargs*)

setLazyLoadGridLatLon(*lazyLoadGridLatLon*)

ThriftClient

class awips.ThriftClient.**ThriftClient**(*host*, *port=9581*, *uri='/services'*)

sendRequest(*request*, *uri='/thrift'*)

exception awips.ThriftClient.**ThriftRequestException**(*value*)

TimeUtil

awips.TimeUtil.**determineDrtOffset**(*timeStr*)

awips.TimeUtil.**makeTime**(*timeStr*)

RadarCommon

awips.RadarCommon.**encode_dep_vals**(*depVals*)

awips.RadarCommon.**encode_radial**(*azVals*)

awips.RadarCommon.**encode_thresh_vals**(*threshVals*)

awips.RadarCommon.**get_data_type**(*azdat*)
Get the radar file type (radial or raster).

Args: azdat: Boolean.

Returns: Radial or raster.

awips.RadarCommon.**get_datetime_str**(*record*)
Get the datetime string for a record.

Args: record: the record to get data for.

Returns: datetime string.

awips.RadarCommon.**get_hdf5_data**(*idra*)

awips.RadarCommon.**get_header**(*record*, *headerFormat*, *xLen*, *yLen*, *azdat*, *description*)

IFPClient

class awips.gfe.IFPClient.**IFPClient**(*host*, *port*, *user*, *site=None*, *progName=None*)

commitGrid(*request*)

getGridInventory(*parmID*)

getParmList(*pid*)

getSelectTR(*name*)

getSiteID()

DateTimeConverter

awips.DateTimeConverter.**constructTimeRange**(**args*)
Builds a python dynamicserialize TimeRange object from the given arguments.

Args:

args*: must be a TimeRange or a pair of objects that can be converted to a datetime via convertToDateTime().

Returns: A TimeRange.

awips.DateTimeConverter.**convertToDateTime**(*timeArg*)
Converts the given object to a python datetime object. Supports native python representations like datetime and struct_time, but also the dynamicserialize types like Date and Timestamp. Raises TypeError if no conversion can be performed.

Args: timeArg: a python object representing a date and time. Supported types include datetime, struct_time, float, int, long and the dynamicserialize types Date and Timestamp.

Returns: A datetime that represents the same date/time as the passed in object.

CombinedTimeQuery

awips.dataaccess.CombinedTimeQuery.**getAvailableTimes**(*request*, *refTimeOnly=False*)

- genindex

Available Data Types

satellite

- 2-D NumPy Array
- returned by: awips.dataaccess.DataAccessLayer.getGridData(request, times=[])
- example:

```
# Contrust a full satellite product tree
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu)
request = DataAccessLayer.newDataRequest("satellite")
creatingEntities = DataAccessLayer.getIdentifierValues(request, "creatingEntity")
for entity in creatingEntities:
    print(entity)
    request = DataAccessLayer.newDataRequest("satellite")
    request.addIdentifier("creatingEntity", entity)
    availableSectors = DataAccessLayer.getAvailableLocationNames(request)
    availableSectors.sort()
    for sector in availableSectors:
        print(" - " + sector)
        request.setLocationNames(sector)
        availableProducts = DataAccessLayer.getAvailableParameters(request)
        availableProducts.sort()
        for product in availableProducts:
            print("    - " + product)
```

—

binlightning

- Shapely Point:

```
POINT (-65.65293884277344 -16.94915580749512)
```

- returned by: awips.dataaccess.DataAccessLayer.getGeometryData(request, times=[])
- example (GLM):

```
request = DataAccessLayer.newDataRequest("binlightning")
request.addIdentifier("source", "GLMgr")
request.setParameters("intensity")
times = DataAccessLayer.getAvailableTimes(request)
response = DataAccessLayer.getGeometryData(request, times[-10:-1])
for ob in response:
    geom = ob.getGeometry()
```

—

grid

- 2-D NumPy Array
- returned by: awips.dataaccess.DataAccessLayer.getGridData(request, times=[])
- example:

```
request = DataAccessLayer.newDataRequest()
request.setDatatype("grid")
request.setLocationNames("RAP13")
request.setParameters("T")
request.setLevels("2.0FHAG")
cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)
fcstRun = DataAccessLayer.getForecastRun(cycles[-1], times)
response = DataAccessLayer.getGridData(request, [fcstRun[-1]])
for grid in response:
    data = grid.getRawData()
    lons, lats = grid.getLatLonCoords()
```

—

warning

- Shapely MultiPolygon, Polygon:

```
MULTIPOLYGON ((-92.092348410 46.782322971, ..., -92.092348410 46.782322971),
              (-90.948581075 46.992865960, ..., -90.948581075 46.992865960),
               ...
              (-92.274543999 46.652773000, ..., -92.280511999 46.656933000),
              (-92.285491999 46.660741000, ..., -92.285491999 46.660741000))
```

- returned by: awips.dataaccess.DataAccessLayer.getGeometryData(request, times=[])
- example:

```
request = DataAccessLayer.newDataRequest()
request.setDatatype("warning")
request.setParameters('phensig')
times = DataAccessLayer.getAvailableTimes(request)
response = DataAccessLayer.getGeometryData(request, times[-50:-1])
for ob in response:
    poly = ob.getGeometry()
    site = ob.getLocationName()
    pd   = ob.getDataTime().getValidPeriod()
    ref  = ob.getDataTime().getRefTime()
```

—

radar

- 2-D NumPy Array
- returned by: awips.dataaccess.DataAccessLayer.getGridData(request, times=[])
- also returned by: RadarCommon.get_hdf5_data(idra)
- example:

```
request = DataAccessLayer.newDataRequest("radar")
request.setLocationNames("kmhx")
request.setParameters("Digital Hybrid Scan Refl")
availableLevels = DataAccessLayer.getAvailableLevels(request)
```

(continues on next page)

(continued from previous page)

```
times = DataAccessLayer.getAvailableTimes(request)
response = DataAccessLayer.getGridData(request, [times[-1]])
for image in response:
    data = image.getRawData()
    lons, lats = image.getLatLonCoords()
```

Data Plotting Examples

AWIPS Grids and Cartopy

Notebook

A simple example of requesting and plotting AWIPS grids with Matplotlib and Cartopy.

```
from awips.dataaccess import DataAccessLayer
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
%matplotlib inline

DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest()
request.setDatatype("grid")
request.setLocationNames("RAP13")
request.setParameters("T")
request.setLevels("2.0FHAG")
cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)
fcstRun = DataAccessLayer.getForecastRun(cycles[-1], times)
response = DataAccessLayer.getGridData(request, [fcstRun[0]])
grid = response[0]
data = grid.getRawData()
lons, lats = grid.getLatLonCoords()
bbox = [lons.min(), lons.max(), lats.min(), lats.max()]

def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(16, 9),
            subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax
```

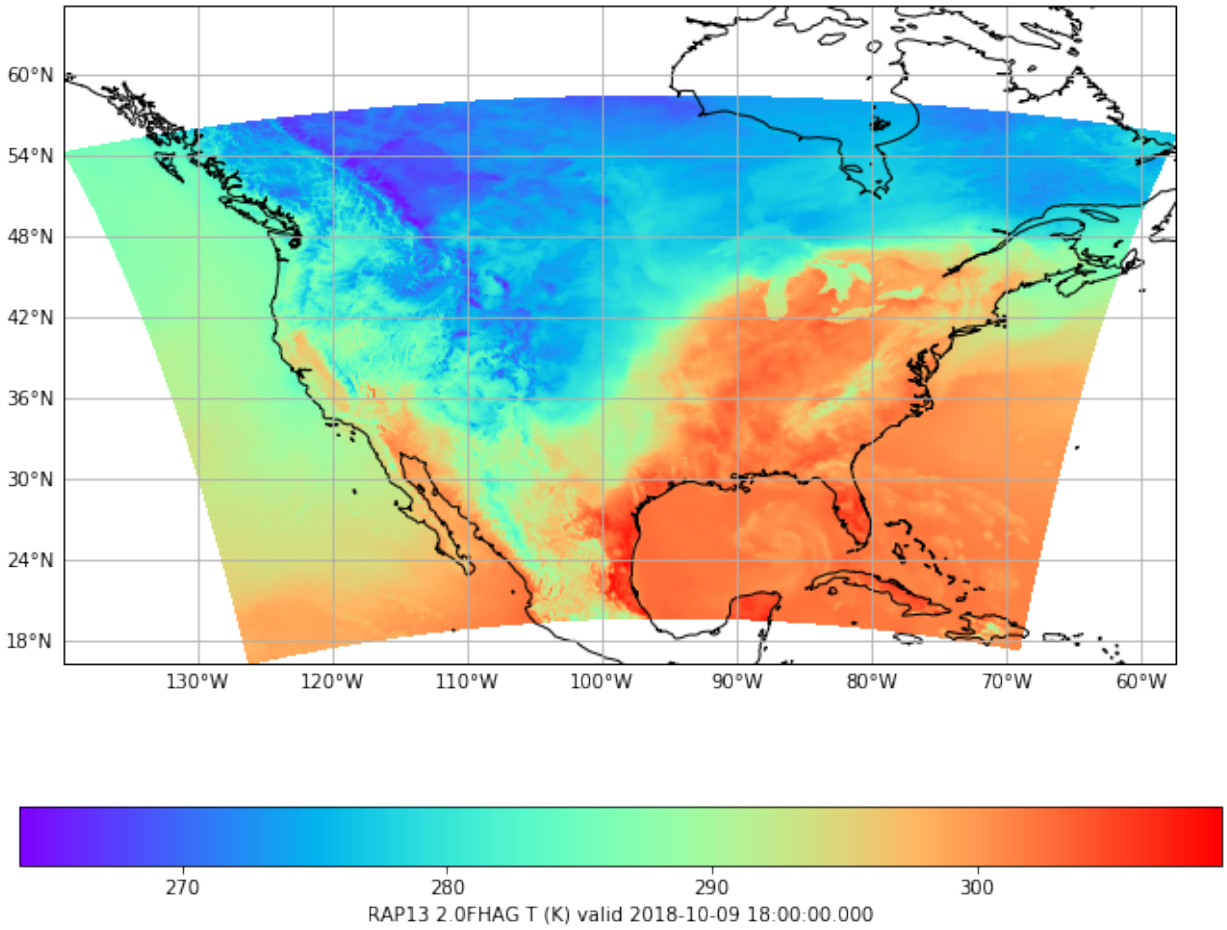
with pcolormesh

```
cmap = plt.get_cmap('rainbow')
fig, ax = make_map(bbox=bbox)
cs = ax.pcolormesh(lons, lats, data, cmap=cmap)
cbar = fig.colorbar(cs, shrink=0.7, orientation='horizontal')
cbar.set_label(grid.getLocationName() +" "+ grid.getLevel() + " " \
```

(continues on next page)

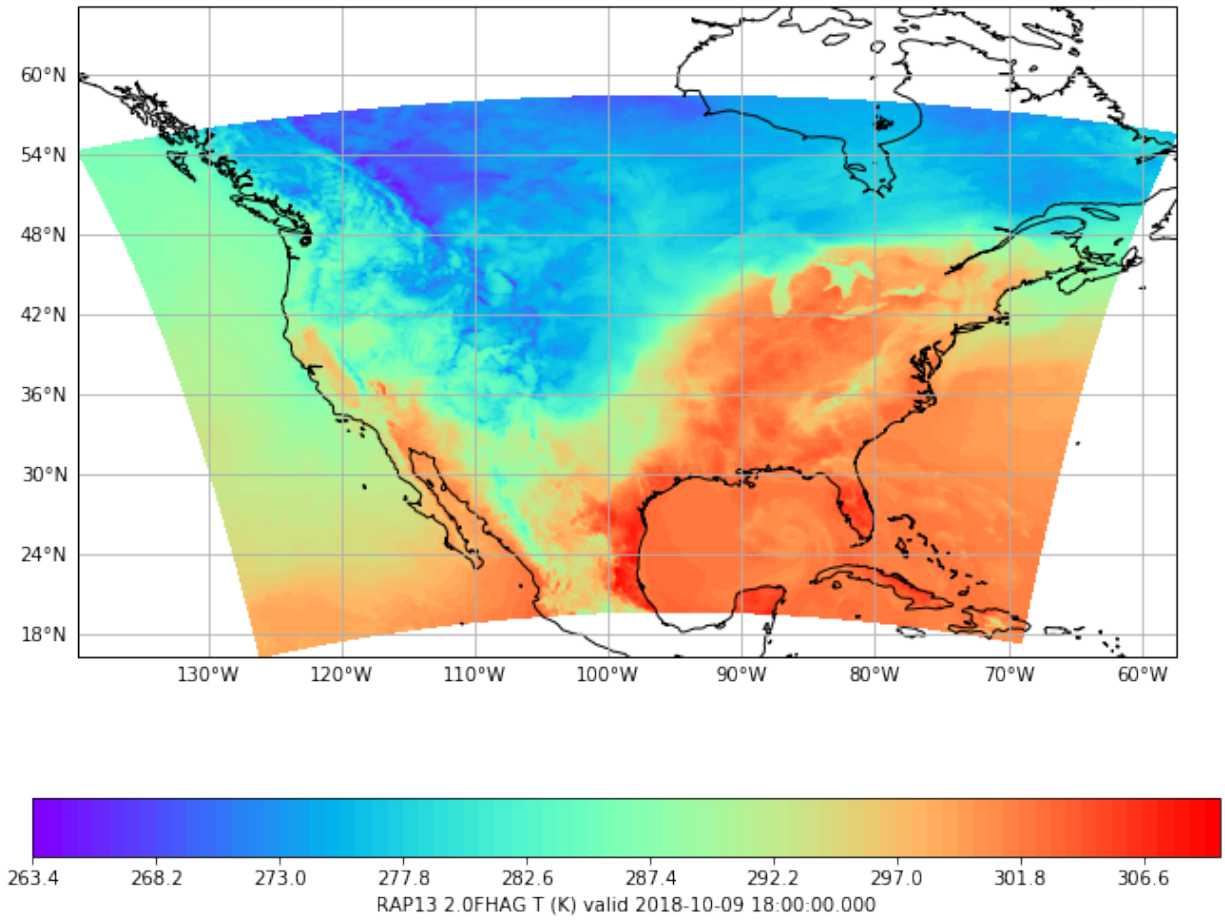
(continued from previous page)

```
                + grid.getParameter() + " (" + grid.getUnit() + ") " \
                + "valid " + str(grid.getDataTime().getRefTime()))
```



with contourf

```
fig2, ax2 = make_map(bbox=bbox)
cs2 = ax2.contourf(lons, lats, data, 80, cmap=cmap,
                   vmin=data.min(), vmax=data.max())
cbar2 = fig2.colorbar(cs2, shrink=0.7, orientation='horizontal')
cbar2.set_label(grid.getLocationName() +" "+ grid.getLevel() + " " \
                + grid.getParameter() + " (" + grid.getUnit()+ ") " \
                + "valid " + str(grid.getDataTime().getRefTime()))
```



Colored Surface Temperature Plot

Notebook

```
%matplotlib inline
```

This exercise creates a colored temperature plot for North America using AWIPS METAR observations (datatype *obs*), similar to existing products in GEMPAK and CAVE.

```
from awips.dataaccess import DataAccessLayer
from dynamicserialize.dstypes.com.raytheon.uf.common.time import TimeRange
from datetime import datetime, timedelta
import numpy as np
import cartopy.crs as ccrs
import warnings
import matplotlib.pyplot as plt
from cartopy.feature import ShapelyFeature
from shapely.geometry import Polygon
from metpy.plots import StationPlot

# CONUS bounding box and envelope geometry
bbox=[-120, -70, 15, 55]
envelope = Polygon([(bbox[0],bbox[2]),(bbox[0],bbox[3]),
```

(continues on next page)

(continued from previous page)

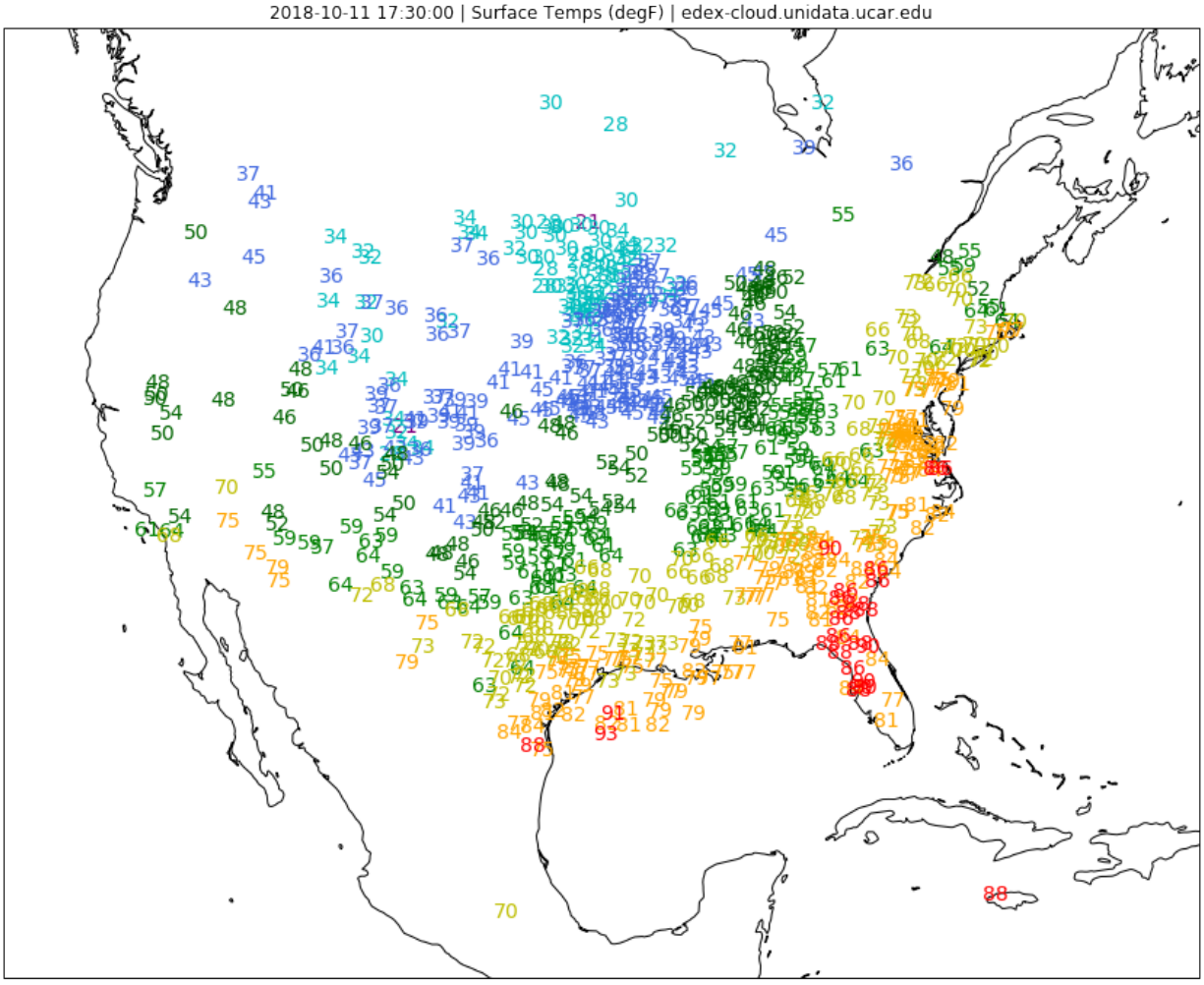
```
                    (bbox[1], bbox[3]),(bbox[1],bbox[2]),
                    (bbox[0],bbox[2])])

# New obs request
edexServer = "edex-cloud.unidata.ucar.edu"
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest("obs", envelope=envelope)
single_value_params = ["timeObs", "stationName", "longitude", "latitude",
                       "temperature", "dewpoint", "windDir",
                       "windSpeed", "seaLevelPress"]
multi_value_params = ["presWeather", "skyCover", "skyLayerBase"]
params = single_value_params + multi_value_params
request.setParameters(*(params))

# Get records from the last 15 minutes
lastHourDateTime = datetime.utcnow() - timedelta(minutes = 15)
start = lastHourDateTime.strftime('%Y-%m-%d %H:%M:%S')
end = datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')

beginRange = datetime.strptime( start , "%Y-%m-%d %H:%M:%S")
endRange = datetime.strptime( end , "%Y-%m-%d %H:%M:%S")
timerange = TimeRange(beginRange, endRange)
# Get response
response = DataAccessLayer.getGeometryData(request,timerange)
obs = DataAccessLayer.getMetarObs(response)

print("Found " + str(len(response)) + " total records")
print("Using " + str(len(obs['temperature'])) + " temperature records")

# Create a station plot pointing to an Axes to draw on as well as the location of␣
↪points

lats = obs['latitude']
lons = obs['longitude']

thresholds = {
    '15': 'purple',
    '25': 'c',
    '35': 'royalblue',
    '45': 'darkgreen',
    '55': 'green',
    '65': 'y',
    '75': 'orange',
    '85': 'red'
}

fig, ax = plt.subplots(figsize=(16,12),subplot_kw=dict(projection=ccrs.
↪LambertConformal()))
ax.set_extent(bbox)
ax.coastlines(resolution='50m')
ax.set_title(str(response[-1].getDataTime()) + " | Surface Temps (degF) | " +␣
↪edexServer)

# Suppress nan masking warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)

for x, value in thresholds.items():
```

(continues on next page)

(continued from previous page)

```
    tair = np.array(obs['temperature'], dtype=float)
    tair[tair == -9999.0] = 'nan'
    tair = (tair*1.8)+32
    if x==max(thresholds):
        tair[(tair < int(x))] = 'nan'
    elif x==min(thresholds):
        tair[(tair >= int(x)+10)] = 'nan'
    else:
        tair[(tair < int(x))] = 'nan'
        tair[(tair >= int(x)+10)] = 'nan'

    stationplot = StationPlot(ax, lons, lats, transform=ccrs.PlateCarree(),
                              fontsize=14)
    stationplot.plot_parameter('C', tair, color=thresholds[str(x)])
```

```
Found 10692 total records
Using 872 temperature records
```



Forecast Model Vertical Sounding

Notebook

The ModelSounding class allows us to create a vertical sounding through any available AWIPS model with isobaric levels.

- A Shapely Point geometry is used to select longitude and latitude: from shapely.geometry import Point point = Point(-104.67,39.87)
- Parameters `['T','DpT','uW','vW']` are requested for all isobaric levels available for the selected model.
- There is a single-record query performed for `level = "0.0FHAG"` to determine the surface pressure level.
- Pay attention to units when switching models. This notebook was written for the NAM 40km AWIPS model where temperature and dewpoint are returned as Kelvin and wind components as m/s.

```
%matplotlib inline
from awips.dataaccess import DataAccessLayer, ModelSounding
from awips import ThriftClient
import matplotlib.pyplot as plt
import numpy as np
from metpy.plots import SkewT, Hodograph
from metpy.units import units
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
from math import sqrt
from datetime import datetime, timedelta
from shapely.geometry import Point, Polygon
import shapely.wkb
import timeit
model="NAM40"
parms = ['T','DpT','uW','vW']
server = 'edex-cloud.unidata.ucar.edu'
DataAccessLayer.changeEDEXHost(server)

# note the order is LON,lat and not lat,LON
point = Point(-104.67,39.87)

inc = 0.005
bbox=[point.y-inc, point.y+inc, point.x-inc, point.x+inc]
polygon = Polygon([(bbox[0],bbox[2]),(bbox[0],bbox[3]),
                   (bbox[1],bbox[3]),(bbox[1],bbox[2]),
                   (bbox[0],bbox[2])])

# Get latest forecast cycle run
timeReq = DataAccessLayer.newDataRequest("grid")
timeReq.setLocationNames(model)
cycles = DataAccessLayer.getAvailableTimes(timeReq, True)
times = DataAccessLayer.getAvailableTimes(timeReq)
fcstRun = DataAccessLayer.getForecastRun(cycles[-2], times)

print("Using " + model + " forecast time " + str(fcstRun[0]))
```

```
Using NAM40 forecast time 2018-10-15 12:00:00
```

```
p,t,d,u,v = [],[],[],[],[]
use_parms = ['T','DpT','uW','vW','P']
use_level = "0.0FHAG"
```

(continues on next page)

(continued from previous page)

```
sndObject = ModelSounding.getSounding(model, use_parms,
                                      ["0.0FHAG"], point, timeRange=[fcstRun[0]])
if len(sndObject) > 0:
    for time in sndObject._dataDict:
        p.append(float(sndObject._dataDict[time][use_level]['P']))
        t.append(float(sndObject._dataDict[time][use_level]['T']))
        d.append(float(sndObject._dataDict[time][use_level]['DpT']))
        u.append(float(sndObject._dataDict[time][use_level]['uW']))
        v.append(float(sndObject._dataDict[time][use_level]['vW']))
    print("Found surface record at " + "%.1f" % p[0] + "MB")
else:
    raise ValueError("sndObject returned empty for query ["
                     + ', '.join(str(x) for x in (model, use_parms, point, use_level))␣
↪+"]")
```

```
Found surface record at 836.4MB
```

```
# Get isobaric levels with our requested parameters
levelReq = DataAccessLayer.newDataRequest("grid", envelope=point)
levelReq.setLocationNames(model)
levelReq.setParameters('T','DpT','uW','vW')
availableLevels = DataAccessLayer.getAvailableLevels(levelReq)

# Clean levels list of unit string (MB, FHAG, etc.)
levels = []
for lvl in availableLevels:
    name=str(lvl)
    if 'MB' in name and '_' not in name:
        # If this level is above (less than in mb) our 0.0FHAG record
        if float(name.replace('MB','')) < p[0]:
            levels.append(lvl)

# Get Sounding
sndObject = ModelSounding.getSounding(model, parms, levels, point,
                        timeRange=[fcstRun[0]])

if not len(sndObject) > 0:
    raise ValueError("sndObject returned empty for query ["
                     + ', '.join(str(x) for x in (model, parms, point, levels)) +"]")

for time in sndObject._dataDict:
    for lvl in sndObject._dataDict[time].levels():
        for parm in sndObject._dataDict[time][lvl].parameters():
            if parm == "T":
                t.append(float(sndObject._dataDict[time][lvl][parm]))
            elif parm == "DpT":
                d.append(float(sndObject._dataDict[time][lvl][parm]))
            elif parm == 'uW':
                u.append(float(sndObject._dataDict[time][lvl][parm]))
            elif parm == 'vW':
                v.append(float(sndObject._dataDict[time][lvl][parm]))
            else:
                print("WHAT IS THIS")
                print(sndObject._dataDict[time][lvl][parm])
        # Pressure is our requested level rather than a returned parameter
        p.append(float(lvl.replace('MB','')))
```

(continues on next page)

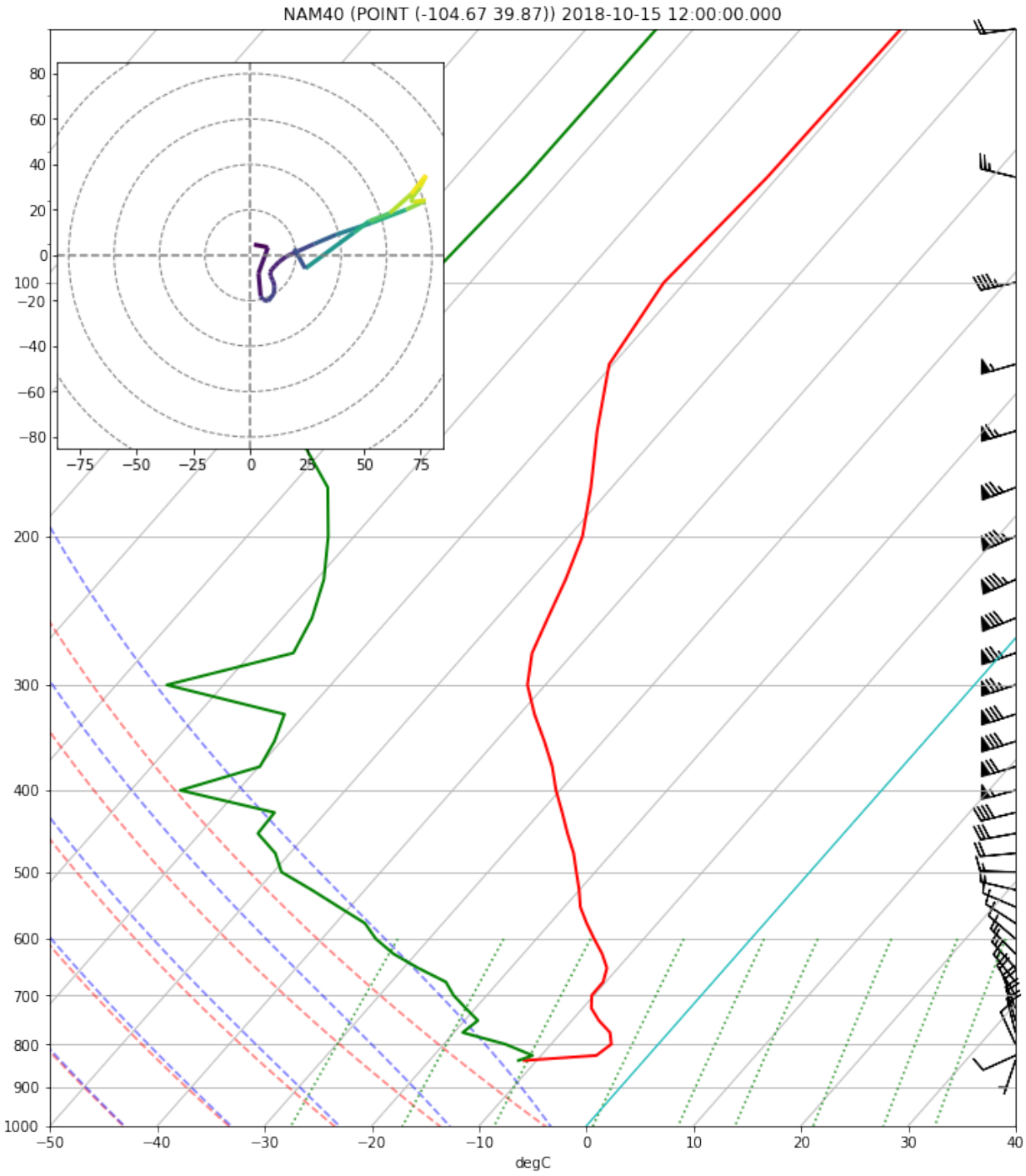
(continued from previous page)

```
# convert to numpy.array()
p = np.array(p, dtype=float)
t = (np.array(t, dtype=float) - 273.15) * units.degC
d = (np.array(d, dtype=float) - 273.15) * units.degC
u = (np.array(u, dtype=float) * units('m/s')).to('knots')
v = (np.array(v, dtype=float) * units('m/s')).to('knots')
w = np.sqrt(u**2 + v**2)

print("Using " + str(len(levels)) + " levels between " +
      str("%.1f" % max(p)) + " and " + str("%.1f" % min(p)) + "MB")
```

```
Using 32 levels between 836.4 and 50.0MB
```

Skew-T/Log-P

```
plt.rcParams['figure.figsize'] = (12, 14)

# Skew-T
skew = SkewT(rotation=45)
skew.plot(p, t, 'r', linewidth=2)
skew.plot(p, d, 'g', linewidth=2)
skew.plot_barbs(p, u, v)
skew.plot_dry_adiabats()
skew.plot_moist_adiabats()
skew.plot_mixing_lines(linestyle=':')

skew.ax.set_ylim(1000, np.min(p))
skew.ax.set_xlim(-50, 40)

# Title
plt.title( model + " (" + str(point) + ") " + str(time.getRefTime()))

# Hodograph
ax_hod = inset_axes(skew.ax, '40%', '40%', loc=2)
h = Hodograph(ax_hod, component_range=max(w.magnitude))
h.add_grid(increment=20)
h.plot_colormapped(u, v, w)

# Dotted line at 0C isotherm
l = skew.ax.axvline(0, color='c', linestyle='-', linewidth=1)

plt.show()
```



Model Sounding Comparison

```
models = ["CMC", "GFS20", "NAM40"]
parms = ['T','DpT','uW','vW']

for modelName in models:
    timeReq = DataAccessLayer.newDataRequest("grid")
```

(continues on next page)

(continued from previous page)

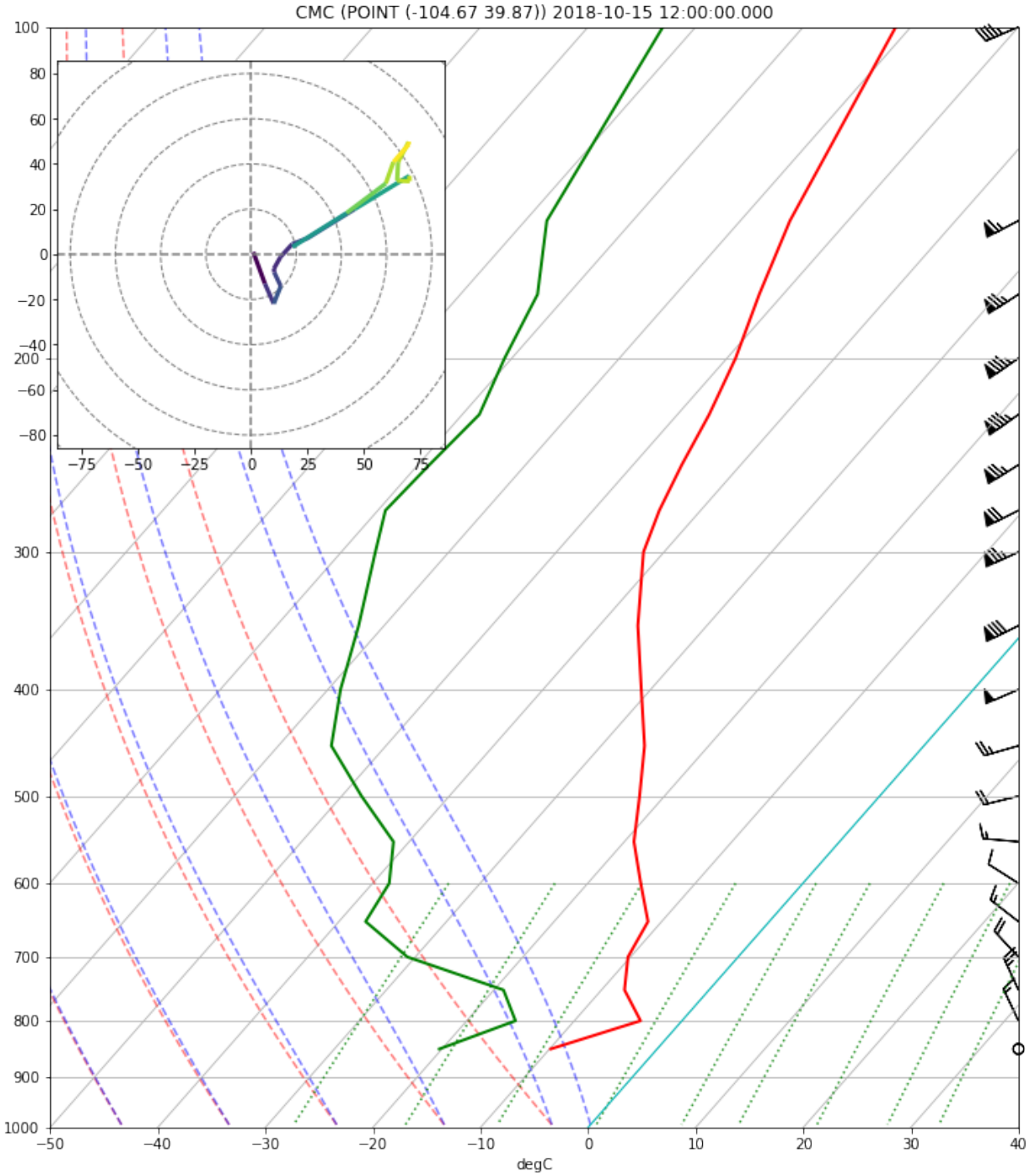
```
    timeReq.setLocationNames(modelName)
    cycles = DataAccessLayer.getAvailableTimes(timeReq, True)
    times = DataAccessLayer.getAvailableTimes(timeReq)
    fcstRun = DataAccessLayer.getForecastRun(cycles[-1], times)
    print("Using " + modelName + " forecast time " + str(fcstRun[0]))

    p,t,d,u,v = [],[],[],[],[]
    use_parms = ['T','DpT','uW','vW','P']
    use_level = "0.0FHAG"

    sndObject = ModelSounding.getSounding(modelName, use_parms,
                                          [use_level], point, timeRange=[fcstRun[0]])
    if len(sndObject) > 0:
        for time in sndObject._dataDict:
            p.append(float(sndObject._dataDict[time][use_level]['P']))
            t.append(float(sndObject._dataDict[time][use_level]['T']))
            d.append(float(sndObject._dataDict[time][use_level]['DpT']))
            u.append(float(sndObject._dataDict[time][use_level]['uW']))
            v.append(float(sndObject._dataDict[time][use_level]['vW']))
        print("Found surface record at " + "%.1f" % p[0] + "MB")
    else:
        raise ValueError("sndObject returned empty for query ["
                         + ', '.join(str(x) for x in (modelName, use_parms, point, use_
↪level)) +"]")

    # Get isobaric levels with our requested parameters
    levelReq = DataAccessLayer.newDataRequest("grid", envelope=point)
    levelReq.setLocationNames(modelName)
    levelReq.setParameters('T','DpT','uW','vW')
    availableLevels = DataAccessLayer.getAvailableLevels(levelReq)
    # Clean levels list of unit string (MB, FHAG, etc.)
    levels = []
    for lvl in availableLevels:
        name=str(lvl)
        if 'MB' in name and '_' not in name:
            # If this level is above (less than in mb) our 0.0FHAG record
            if float(name.replace('MB','')) < p[0]:
                levels.append(lvl)

    # Get Sounding
    sndObject = ModelSounding.getSounding(modelName, parms, levels, point,
                              timeRange=[fcstRun[0]])
    if not len(sndObject) > 0:
        raise ValueError("sndObject returned empty for query ["
                         + ', '.join(str(x) for x in (modelName, parms, point,␣
↪levels)) +"]")
    for time in sndObject._dataDict:
        for lvl in sndObject._dataDict[time].levels():
            for parm in sndObject._dataDict[time][lvl].parameters():
                if parm == "T":
                    t.append(float(sndObject._dataDict[time][lvl][parm]))
                elif parm == "DpT":
                    d.append(float(sndObject._dataDict[time][lvl][parm]))
                elif parm == 'uW':
                    u.append(float(sndObject._dataDict[time][lvl][parm]))
                elif parm == 'vW':
                    v.append(float(sndObject._dataDict[time][lvl][parm]))
```

(continues on next page)

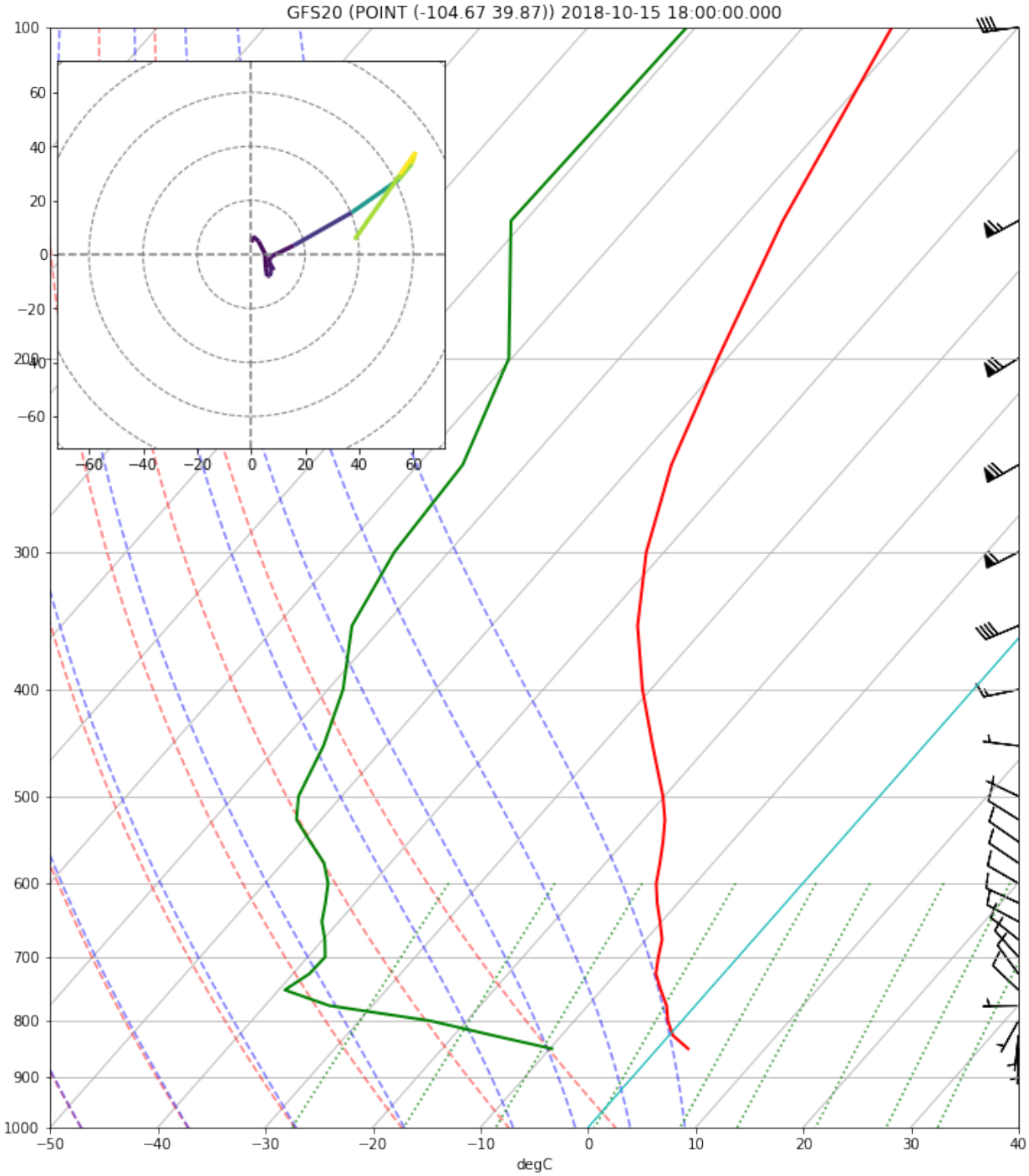
(continued from previous page)

```
                else:
                    print("WHAT IS THIS")
                    print(sndObject._dataDict[time][lvl][parm])
            # Pressure is our requested level rather than a returned parameter
            p.append(float(lvl.replace('MB','')))

    # convert to numpy.array()
    p = np.array(p, dtype=float)
    t = (np.array(t, dtype=float) - 273.15) * units.degC
    d = (np.array(d, dtype=float) - 273.15) * units.degC
    u = (np.array(u, dtype=float) * units('m/s')).to('knots')
    v = (np.array(v, dtype=float) * units('m/s')).to('knots')
    w = np.sqrt(u**2 + v**2)

    print("Using " + str(len(levels)) + " levels between " +
          str("%.1f" % max(p)) + " and " + str("%.1f" % min(p)) + "MB")

    # Skew-T
    plt.rcParams['figure.figsize'] = (12, 14)
    skew = SkewT(rotation=45)
    skew.plot(p, t, 'r', linewidth=2)
    skew.plot(p, d, 'g', linewidth=2)
    skew.plot_barbs(p, u, v)
    skew.plot_dry_adiabats()
    skew.plot_moist_adiabats()
    skew.plot_mixing_lines(linestyle=':')
    skew.ax.set_ylim(1000, 100)
    skew.ax.set_xlim(-50, 40)
    # Title
    plt.title( modelName + " (" + str(point) + ") " + str(time.getRefTime()))
    # Hodograph
    ax_hod = inset_axes(skew.ax, '40%', '40%', loc=2)
    h = Hodograph(ax_hod, component_range=max(w.magnitude))
    h.add_grid(increment=20)
    h.plot_colormapped(u, v, w)
    # Dotted line at 0C isotherm
    l = skew.ax.axvline(0, color='c', linestyle='-', linewidth=1)
    plt.show()
```

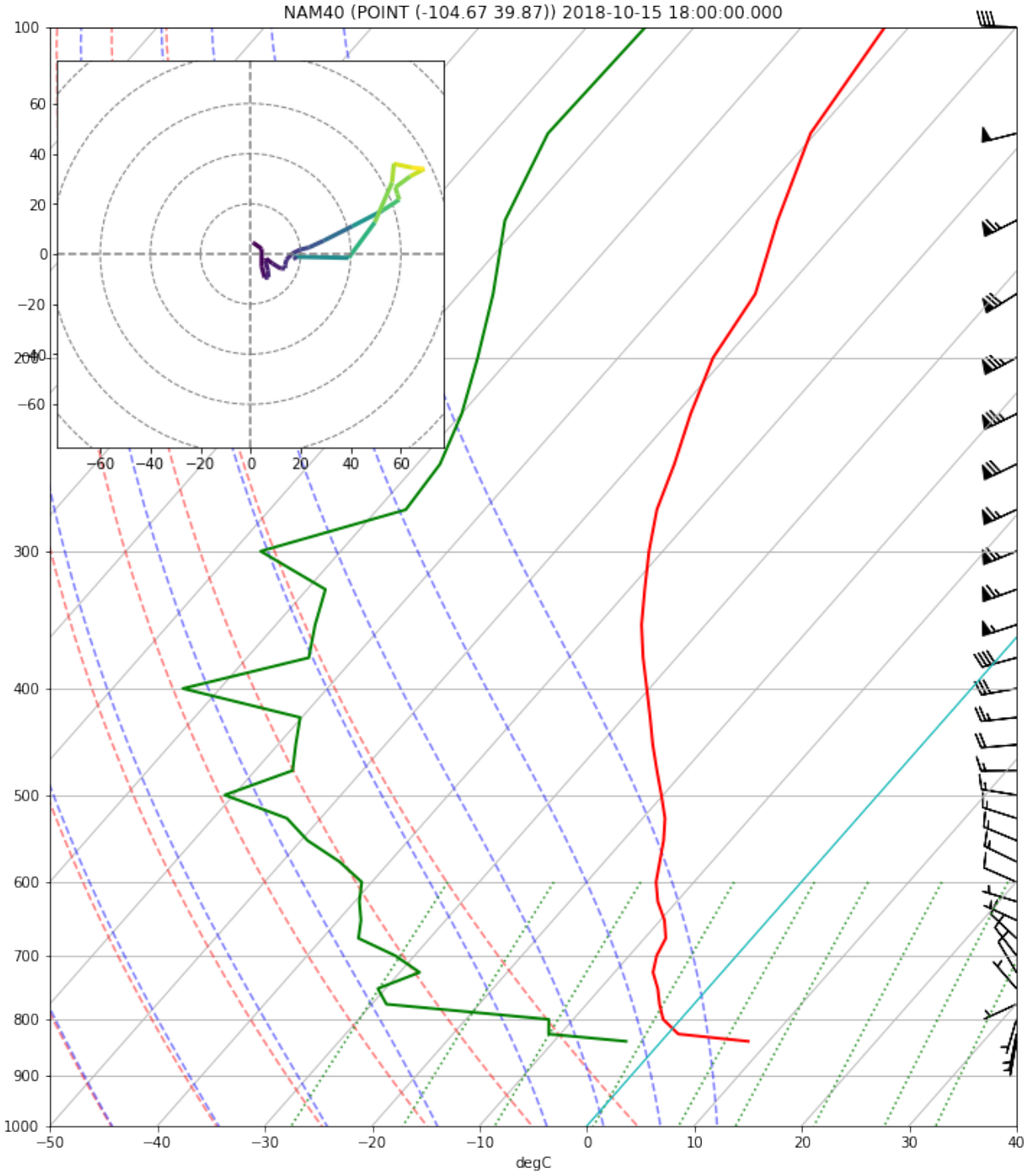
```
Using CMC forecast time 2018-10-15 12:00:00
Found surface record at 848.6MB
Using 19 levels between 848.6 and 50.0MB
```




```
Using GFS20 forecast time 2018-10-15 18:00:00
Found surface record at 848.1MB
Using 22 levels between 848.1 and 100.0MB
```




```
Using NAM40 forecast time 2018-10-15 18:00:00
Found surface record at 837.7MB
Using 32 levels between 837.7 and 50.0MB
```



GOES Geostationary Lightning Mapper

Notebook

The Geostationary Lightning Mapper, or GLM, on board GOES-R Series spacecraft, is the first operational lightning

mapper flown in geostationary orbit. GLM detects the light emitted by lightning at the tops of clouds day and night and collects information such as the frequency, location and extent of lightning discharges. The instrument measures total lightning, both in-cloud and cloud-to-ground, to aid in forecasting developing severe storms and a wide range of high-impact environmental phenomena including hailstorms, microburst winds, tornadoes, hurricanes, flash floods, snowstorms and fires.

AWIPS GLM point data are available in three formats

- `GLMev` Events
- `GLMfl` Flashes
- `GLMgr` Groups

and with seven attributes:

- height
- intensity
- msgType
- pulseCount
- pulseIndex
- sensorCount
- strikeType

GLM Sources and Parameters

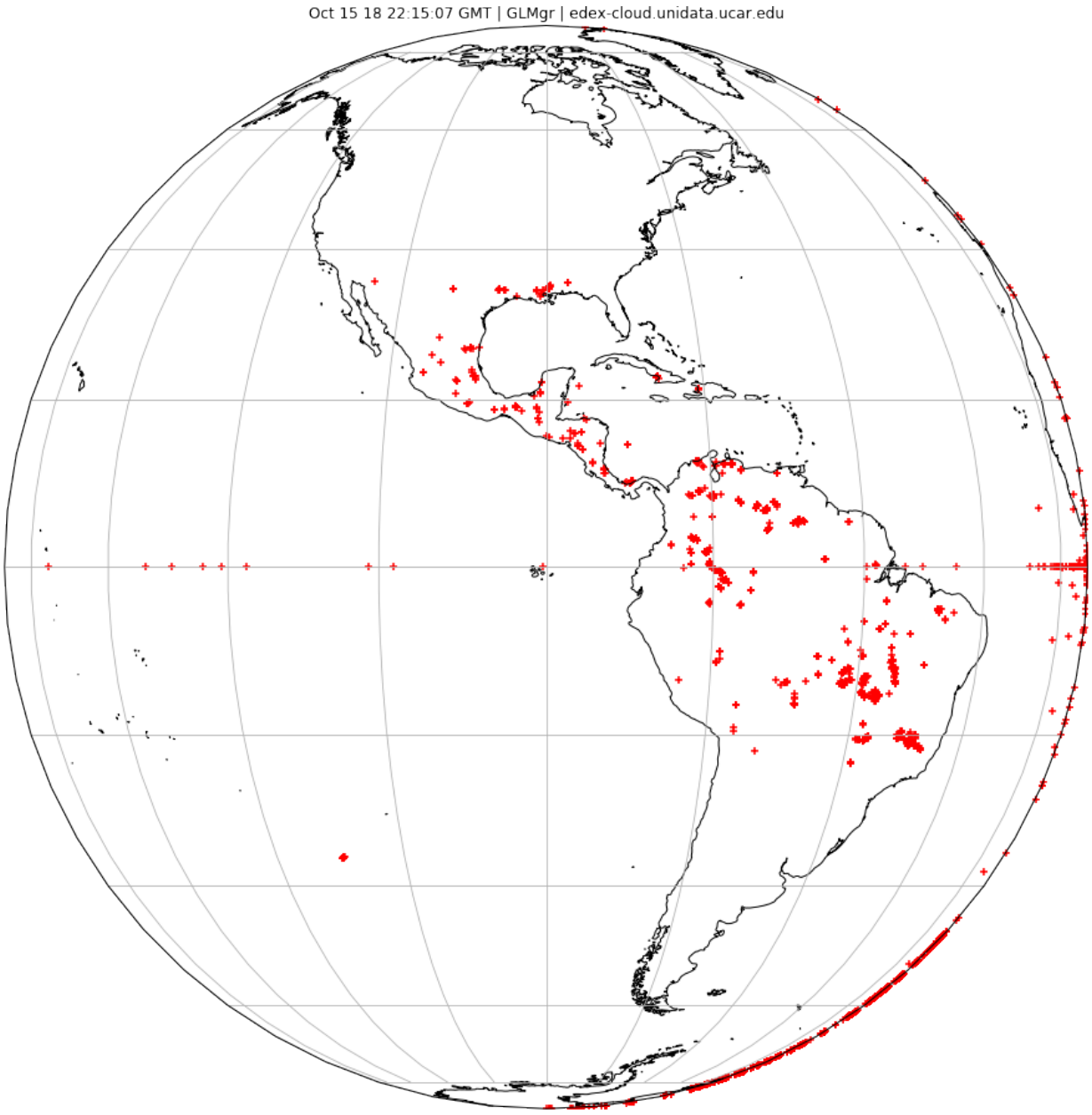
```
from awips.dataaccess import DataAccessLayer
import cartopy.crs as ccrs
import cartopy.feature as cfeat
import matplotlib.pyplot as plt
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
import numpy as np
import datetime

%matplotlib inline

# Create an EDEX data request
edexServer = "edex-cloud.unidata.ucar.edu"
datatype = "binlightning"
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest(datatype)

# Show available sources
sources = DataAccessLayer.getIdentifierValues(request, "source")
print("available sources:")
print(list(sources))
print("")
availableParms = DataAccessLayer.getAvailableParameters(request)
availableParms.sort()
print("available parameters:")
print(list(availableParms))
```

```
available sources:
['GLMgr', 'GLMfl', 'GLMev']

available parameters:
['height', 'intensity', 'msgType', 'pulseCount', 'pulseIndex', 'sensorCount',
↪'strikeType']
```

```
request.addIdentifier("source", "GLMgr")
request.setParameters("intensity")
times = DataAccessLayer.getAvailableTimes(request)
response = DataAccessLayer.getGeometryData(request, [times[-1]])
```

```
# Plot markers
fig, ax = plt.subplots(figsize=(16,16),subplot_kw=dict(projection=ccrs.
↪Orthographic(central_longitude=-90.0)))
ax.coastlines(resolution='50m')
ax.gridlines()
ax.scatter([point.x for point in glm_points],
       [point.y for point in glm_points],
       transform=ccrs.Geodetic(),marker="+",facecolor='red')
ax.set_title(str(response[-1].getDataTime().getRefTime()) + " | " + ob.getAttribute(
↪'source') + " | " + edexServer)
```

```
Text(0.5,1,'Oct 15 18 22:15:07 GMT | GLMgr | edex-cloud.unidata.ucar.edu')
```



Grid Levels and Parameters

Notebook

This example covers the callable methods of the Python AWIPS DAF when working with gridded data. We start with a connection to an EDEX server, then query data types, then grid names, parameters, levels, and other information. Finally the gridded data is plotted for its domain using Matplotlib and Cartopy.

DataAccessLayer.getSupportedDatatypes()

getSupportedDatatypes() returns a list of available data types offered by the EDEX server defined above.

```
from awips.dataaccess import DataAccessLayer
import unittest

DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
dataTypes = DataAccessLayer.getSupportedDatatypes()
dataTypes.sort()
list(dataTypes)
```

```
['acars',
 'binlightning',
 'bufrmosAVN',
 'bufrmosETA',
 'bufrmosGFS',
 'bufrmosHPC',
 'bufrmosLAMP',
 'bufrmosMRF',
 'bufrua',
 'climate',
 'common_obs_spatial',
 'gfe',
 'gfeEditArea',
 'grid',
 'maps',
 'modelsounding',
 'obs',
 'practicewarning',
 'profiler',
 'radar',
 'radar_spatial',
 'satellite',
 'sfcobs',
 'topo',
 'warning']
```

DataAccessLayer.getAvailableLocationNames()

Now create a new data request, and set the data type to **grid** to request all available grids with **getAvailableLocationNames()**

```
request = DataAccessLayer.newDataRequest()
request.setDatatype("grid")
available_grids = DataAccessLayer.getAvailableLocationNames(request)
available_grids.sort()
list(available_grids)
```

```
['CMC',
 'ESTOFS',
 'ETSS',
 'FFG-ALR',
 'FFG-FWR',
 'FFG-KRF',
 'FFG-MSR',
 'FFG-ORN',
 'FFG-PTR',
 'FFG-RHA',
```

(continues on next page)

(continued from previous page)

```
 'FFG-RSA',
 'FFG-STR',
 'FFG-TAR',
 'FFG-TIR',
 'FFG-TUA',
 'FNMOC-FAROP',
 'FNMOC-NCODA',
 'GFS',
 'GFS20',
 'GribModel:9:159:180',
 'HFR-EAST_6KM',
 'HFR-EAST_PR_6KM',
 'HFR-US_EAST_DELAWARE_1KM',
 'HFR-US_EAST_FLORIDA_2KM',
 'HFR-US_EAST_NORTH_2KM',
 'HFR-US_EAST_SOUTH_2KM',
 'HFR-US_EAST_VIRGINIA_1KM',
 'HFR-US_HAWAII_1KM',
 'HFR-US_HAWAII_2KM',
 'HFR-US_HAWAII_6KM',
 'HFR-US_WEST_500M',
 'HFR-US_WEST_CENCAL_2KM',
 'HFR-US_WEST_LOSANGELES_1KM',
 'HFR-US_WEST_LOSOSOS_1KM',
 'HFR-US_WEST_NORTH_2KM',
 'HFR-US_WEST_SANFRAN_1KM',
 'HFR-US_WEST_SOCAL_2KM',
 'HFR-US_WEST_WASHINGTON_1KM',
 'HFR-WEST_6KM',
 'HPCGuide',
 'HPCqpf',
 'HPCqpfNDFD',
 'HRRR',
 'LAMP2p5',
 'MOSGuide',
 'NAM12',
 'NAM40',
 'NCWF',
 'NOHRSC-SNOW',
 'NationalBlend',
 'PROB3HR',
 'QPE-RFC-STR',
 'RAP13',
 'RFCqpf',
 'RTMA',
 'SPCGuide',
 'SeaIce',
 'TPCWindProb',
 'UKMET-MODEL1',
 'URMA25',
 'fnmocWave',
 'nogaps']
```

DataAccessLayer.getAvailableParameters()

After datatype and model name (locationName) are set, you can query all available parameters with **getAvailableParameters()**

```
request.setLocationNames("RAP13")
availableParms = DataAccessLayer.getAvailableParameters(request)
availableParms.sort()
list(availableParms)
```

```
['0to5',
 '2xTP6hr',
 '36SHRMi',
 '50dbzZ',
 'AV',
 'Along',
 'AppT',
 'BLI',
 'BRN',
 'BRNEHIi',
 'BRNSHR',
 'BRNmag',
 'BRNvec',
 'BdEPT06',
 'BlkMag',
 'BlkShr',
 'CAPE',
 'CFRZR',
 'CICEP',
 'CIn',
 'CP',
 'CP1hr',
 'CPr',
 'CPrD',
 'CRAIN',
 'CSNOW',
 'CURU',
 'CXR',
 'CapeStk',
 'Corf',
 'CorfF',
 'CorfFM',
 'CorfM',
 'CritT1',
 'CumNrm',
 'CumShr',
 'DIABi',
 'DivF',
 'DivFn',
 'DivFs',
 'DpD',
 'DpDt',
 'DpT',
 'Dpress',
 'DthDt',
 'EHI',
 'EHI01',
```

(continues on next page)

(continued from previous page)

```
'EHIi',
'EPT',
'EPTA',
'EPTC',
'EPTGrd',
'EPTGrdM',
'EPTs',
'EPVg',
'EPVs',
'EPVt1',
'EPVt2',
'ESP',
'ESP2',
'FRZR12hr',
'FRZRrun',
'FVecs',
'FeatMot',
'FnVecs',
'FsVecs',
'Fzra1',
'Fzra2',
'GH',
'GHxSM',
'GHxSM2',
'Gust',
'HI',
'HI1',
'HI3',
'HI4',
'HIdx',
'HPBL',
'Heli',
'HeliC',
'INV',
'IPLayer',
'Into',
'KI',
'L-I',
'LIsfc2x',
'LM5',
'LM6',
'MAdv',
'MCon',
'MCon2',
'MLLCL',
'MMP',
'MMSP',
'MSFDi',
'MSFi',
'MSFmi',
'MSG',
'MTV',
'Mix1',
'Mix2',
'Mmag',
'MnT',
'MpV',
```

(continues on next page)

(continued from previous page)

```
'MxT',
'NBE',
'NST',
'NST1',
'NST2',
'NetIO',
'OmDiff',
'P',
'PAdv',
'PBE',
'PEC',
'PEC_TT24',
'PFrnt',
'PGrd',
'PGrd1',
'PGrdM',
'PIVA',
'PR',
'PTvA',
'PTyp',
'PVV',
'PW',
'PW2',
'PoT',
'PoTA',
'QPV1',
'QPV2',
'QPV3',
'QPV4',
'REFC',
'RH',
'RH_001_bin',
'RH_002_bin',
'RM5',
'RM6',
'RMGH2',
'RMprop',
'RMprop2',
'RRtype',
'RV',
'Rain1',
'Rain2',
'Rain3',
'Ro',
'SA12hr',
'SA1hr',
'SA24hr',
'SA36hr',
'SA3hr',
'SA48hr',
'SA6hr',
'SAcc',
'SArun',
'SH',
'SHx',
'SLI',
'SNSQ',
```

(continues on next page)

(continued from previous page)

```
'SNW',
'SNWA',
'SRMl',
'SRMlM',
'SRMm',
'SRMmM',
'SRMr',
'SRMrM',
'SSP',
'SSi',
'STP',
'STP1',
'Shear',
'ShrMag',
'SnD',
'Snow1',
'Snow2',
'Snow3',
'SnowT',
'St-Pr',
'StrTP',
'StrmMot',
'SuCP',
'T',
'TAdv',
'TGrd',
'TGrdM',
'TORi',
'TORi2',
'TP',
'TP12hr',
'TP168hr',
'TP1hr',
'TP24hr',
'TP36hr',
'TP3hr',
'TP48hr',
'TP6hr',
'TP72hr',
'TPrun',
'TPx12x6',
'TPx1x3',
'TQIND',
'TShrMi',
'TV',
'TW',
'T_001_bin',
'Tdef',
'Tdend',
'ThGrd',
'Thom5',
'Thom5a',
'Thom6',
'TmDpD',
'Tmax',
'Tmin',
'Topo',
```

(continues on next page)

(continued from previous page)

```
'TotQi',
'Tstk',
'TwMax',
'TwMin',
'Twstk',
'TxSM',
'USTM',
'VAdv',
'VAdvAdvection',
'VGP',
'VSTM',
'Vis',
'WCD',
'WD',
'WEASD',
'WEASD1hr',
'WGS',
'Wind',
'WndChl',
'ageoVC',
'ageoW',
'ageoWM',
'cCape',
'cCin',
'cTOT',
'capeToLvl',
'dCape',
'dGH12',
'dP',
'dP1hr',
'dP3hr',
'dP6hr',
'dPW1hr',
'dPW3hr',
'dPW6hr',
'dT',
'dVAdv',
'dZ',
'defV',
'del2gH',
'df',
'fGen',
'fnD',
'fsD',
'gamma',
'gammaE',
'geoVort',
'geoW',
'geoWM',
'loCape',
'maxEPT',
'minEPT',
'mixRat',
'msl-P',
'muCape',
'pV',
'pVeq',
```

(continues on next page)

(continued from previous page)

```
 'qDiv',
 'qVec',
 'qnVec',
 'qsVec',
 'shWlt',
 'snoRat',
 'snoRatCrocus',
 'snoRatEMCSREF',
 'snoRatOv2',
 'snoRatSPC',
 'snoRatSPCdeep',
 'snoRatSPCsurface',
 'staticCoriolis',
 'staticSpacing',
 'staticTopo',
 'swtIdx',
 'tTOT',
 'tWind',
 'tWindU',
 'tWindV',
 'uFX',
 'uW',
 'uWStk',
 'ulSnoRat',
 'vSmthW',
 'vTOT',
 'vW',
 'vWStk',
 'vertCirc',
 'wDiv',
 'wSp',
 'wSp_001_bin',
 'wSp_002_bin',
 'wSp_003_bin',
 'wSp_004_bin',
 'zAGL']
```

DataAccessLayer.getAvailableLevels()

Selecting **"T"** for temperature.

```
request.setParameters("T")
availableLevels = DataAccessLayer.getAvailableLevels(request)
for lvl in availableLevels:
    print(lvl)
```

```
100.0MB
175.0MB
125.0MB
200.0MB
150.0MB
250.0MB
225.0MB
275.0MB
```

(continues on next page)

(continued from previous page)

```
300.0MB
325.0MB
350.0MB
400.0MB
375.0MB
425.0MB
450.0MB
475.0MB
500.0MB
525.0MB
550.0MB
575.0MB
650.0MB
625.0MB
600.0MB
675.0MB
700.0MB
725.0MB
750.0MB
775.0MB
825.0MB
800.0MB
850.0MB
875.0MB
900.0MB
925.0MB
975.0MB
1000.0MB
0.0SFC
950.0MB
0.0TROP
340.0_350.0K
290.0_300.0K
700.0_600.0MB
700.0_300.0MB
320.0Ke
800.0_750.0MB
0.0_610.0FHAG
60.0TILT
5.3TILT
1000.0_900.0MB
340.0K
1000.0_500.0MB
5500.0_6000.0FHAG
255.0K
255.0_265.0K
3000.0_6000.0FHAG
25.0TILT
2000.0FHAG
0.0_500.0FHAG
1000.0_850.0MB
850.0_250.0MB
280.0_290.0Ke
1524.0FHAG
320.0_330.0K
0.0TILT
150.0_180.0BL
```

(continues on next page)

(continued from previous page)

```
0.0_30.0BL
310.0_320.0Ke
310.0Ke
330.0K
900.0_800.0MB
550.0_500.0MB
2.4TILT
50.0TILT
3500.0FHAG
35.0TILT
12.0TILT
300.0_310.0K
3000.0_12000.0FHAG
0.9TILT
320.0K
400.0_350.0MB
500.0FHAG
750.0_700.0MB
1000.0_400.0MB
345.0K
250.0_260.0K
300.0Ke
290.0Ke
950.0_900.0MB
4572.0FHAG
275.0_285.0Ke
335.0Ke
295.0_305.0Ke
275.0_285.0K
600.0_550.0MB
310.0K
9000.0FHAG
335.0K
1000.0_7000.0FHAG
700.0_500.0MB
9144.0FHAG
325.0_335.0K
2000.0_8000.0FHAG
0.0_609.6FHAG
300.0K
0.0MAXOMEGA
315.0_325.0K
325.0K
340.0Ke
0.0_4000.0FHAG
5000.0_5500.0FHAG
300.0_250.0MB
1.5TILT
335.0_345.0K
2.0FHAG
315.0K
3.4TILT
2500.0FHAG
10000.0FHAG
0.0_2000.0FHAG
7000.0FHAG
0.0_1000.0FHAG
```

(continues on next page)

(continued from previous page)

```
5000.0FHAG
330.0Ke
90.0_120.0BL
500.0_400.0MB
1000.0_1500.0FHAG
305.0K
285.0_295.0Ke
14.0TILT
3000.0_3500.0FHAG
325.0_335.0Ke
2000.0_5000.0FHAG
7620.0FHAG
850.0_800.0MB
120.0_150.0BL
6096.0FHAG
6000.0_7000.0FHAG
2000.0_7000.0FHAG
9000.0_10000.0FHAG
295.0Ke
305.0Ke
30.0_60.0BL
265.0_275.0K
7000.0_8000.0FHAG
3000.0_8000.0FHAG
700.0_650.0MB
1000.0_6000.0FHAG
0.5TILT
450.0_400.0MB
1.8TILT
330.0_340.0K
800.0_700.0MB
850.0_300.0MB
4000.0FHAG
6.0TILT
900.0_850.0MB
3657.6FHAG
0.0_5000.0FHAG
320.0_330.0Ke
8.7TILT
650.0_600.0MB
0.0FHAG
600.0_400.0MB
55.0TILT
270.0_280.0Ke
30.0TILT
310.0_320.0K
1500.0FHAG
1000.0_950.0MB
1.0PV
5500.0FHAG
250.0_200.0MB
1.5PV
500.0_1000.0FHAG
400.0_300.0MB
500.0_100.0MB
1000.0_3000.0FHAG
8000.0FHAG
```

(continues on next page)

(continued from previous page)

```
285.0Ke
290.0K
305.0_315.0K
285.0_295.0K
0.0_2500.0FHAG
925.0_850.0MB
275.0Ke
1500.0_2000.0FHAG
2.0PV
300.0_200.0MB
610.0_40000.0FHAG
260.0_270.0K
0.0_6000.0FHAG
2743.2FHAG
3000.0FHAG
315.0_325.0Ke
600.0_500.0MB
16.7TILT
0.5PV
280.0K
500.0_250.0MB
40.0TILT
1000.0FHAG
3048.0FHAG
400.0_200.0MB
300.0_310.0Ke
270.0_280.0K
1000.0_700.0MB
45.0TILT
850.0_500.0MB
60.0_90.0BL
2500.0_3000.0FHAG
609.6FHAG
0.0_8000.0FHAG
295.0K
4.3TILT
295.0_305.0K
330.0_340.0Ke
270.0K
4000.0_4500.0FHAG
280.0_290.0K
925.0_700.0MB
0.0_1500.0FHAG
260.0K
10.0TILT
3500.0_4000.0FHAG
325.0Ke
285.0K
290.0_300.0Ke
7.5TILT
1828.8FHAG
280.0Ke
500.0_450.0MB
305.0_315.0Ke
250.0K
4500.0FHAG
1250.0FHAG
```

(continues on next page)

(continued from previous page)

```
0.0_10000.0FHAG
4500.0_5000.0FHAG
250.0_350.0K
270.0Ke
275.0K
315.0Ke
500.0_300.0MB
350.0_300.0MB
750.0FHAG
19.5TILT
0.0_3000.0FHAG
2000.0_2500.0FHAG
850.0_700.0MB
350.0K
265.0K
6000.0FHAG
8000.0_9000.0FHAG
2000.0_3000.0FHAG
```

- **0.0SFC** is the Surface level
- **FHAG** stands for Fixed Height Above Ground (in meters)
- **NTAT** stands for Nominal Top of the ATmosphere
- **BL** stands for Boundary Layer, where **0.0_30.0BL** reads as *0-30 mb above ground level*
- **TROP** is the Tropopause level

request.setLevels()

For this example we will use Surface Temperature

```
request.setLevels("2.0FHAG")
```

DataAccessLayer.getAvailableTimes()

- **getAvailableTimes(request, True)** will return an object of *run times* - formatted as `YYYY-MM-DD HH:MM:SS`
- **getAvailableTimes(request)** will return an object of all times - formatted as `YYYY-MM-DD HH:MM:SS (F:ff)`
- **getForecastRun(cycle, times)** will return a DataTime array for a single forecast cycle.

```
cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)
fcstRun = DataAccessLayer.getForecastRun(cycles[-1], times)
list(fcstRun)
```

```
[<DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
```

(continues on next page)

(continued from previous page)

```
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >,
 <DataTime instance: 2018-10-09 18:00:00 >]
```

DataAccessLayer.getGridData()

Now that we have our `request` and DataTime `fcstRun` arrays ready, it's time to request the data array from EDEX.

```
response = DataAccessLayer.getGridData(request, [fcstRun[-1]])
for grid in response:
    data = grid.getRawData()
    lons, lats = grid.getLatLonCoords()
    print('Time :', str(grid.getDataTime()))

print('Model:', str(grid.getLocationName()))
print('Parm :', str(grid.getParameter()))
print('Unit :', str(grid.getUnit()))
print(data.shape)
```

```
Time : 2018-10-09 18:00:00
Model: RAP13
Parm : T
Unit : K
(337, 451)
```

Plotting with Matplotlib and Cartopy

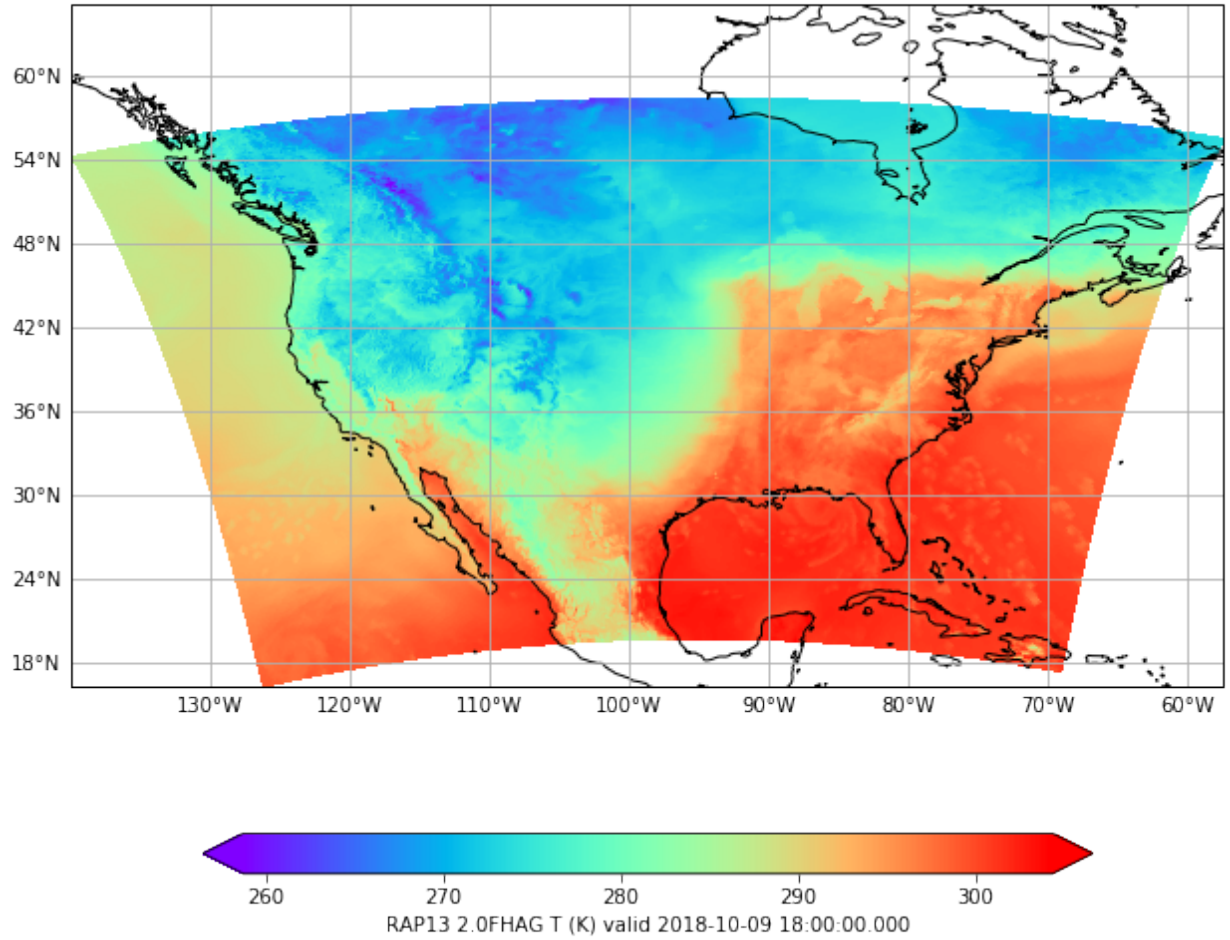
1. pcolormesh

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
import numpy as np
import numpy.ma as ma
from scipy.io import loadmat
def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(16, 9),
                           subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
```

(continues on next page)

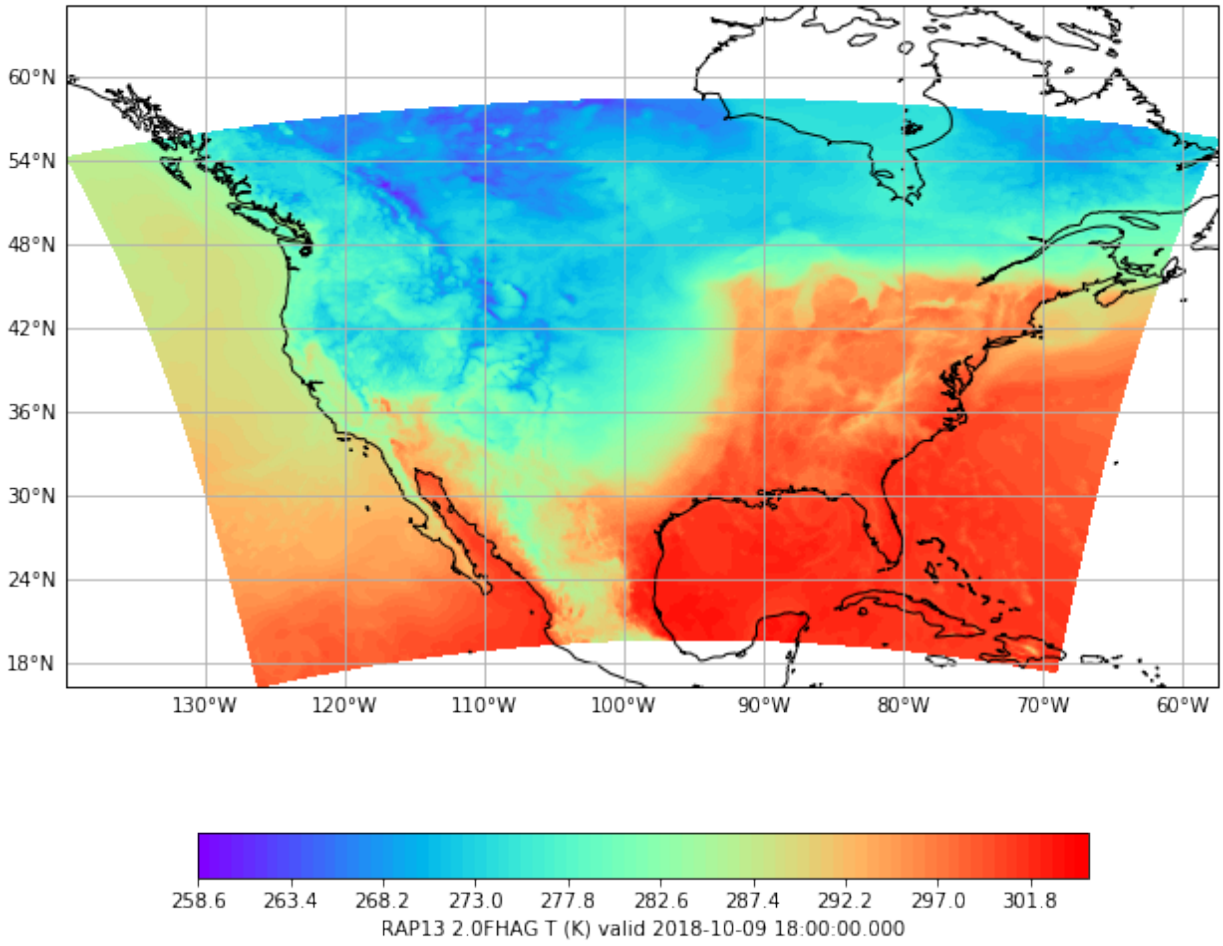
(continued from previous page)

```
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax

cmap = plt.get_cmap('rainbow')
bbox = [lons.min(), lons.max(), lats.min(), lats.max()]
fig, ax = make_map(bbox=bbox)
cs = ax.pcolormesh(lons, lats, data, cmap=cmap)
cbar = fig.colorbar(cs, extend='both', shrink=0.5, orientation='horizontal')
cbar.set_label(grid.getLocationName() +" " + grid.getLevel() + " " \
               + grid.getParameter() + " (" + grid.getUnit() + ") " \
               + "valid " + str(grid.getDataTime().getRefTime()))
```



2. contourf

```
fig2, ax2 = make_map(bbox=bbox)
cs2 = ax2.contourf(lons, lats, data, 80, cmap=cmap,
                   vmin=data.min(), vmax=data.max())
cbar2 = fig2.colorbar(cs2, extend='both', shrink=0.5, orientation='horizontal')
cbar2.set_label(grid.getLocationName() +" " + grid.getLevel() + " " \
               + grid.getParameter() + " (" + grid.getUnit() + ") " \
               + "valid " + str(grid.getDataTime().getRefTime()))
```



METAR Station Plot with MetPy

Notebook

```
%matplotlib inline
```

This exercise creates a METAR plot for North America using AWIPS METAR observations (datatype *obs*) and MetPy.

```
from awips.dataaccess import DataAccessLayer
from dynamicserialize.dstypes.com.raytheon.uf.common.time import TimeRange
from datetime import datetime, timedelta
import numpy as np
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
from metpy.calc import wind_components
from metpy.plots import StationPlot, StationPlotLayout
from metpy.units import units
import warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)

def get_cloud_cover(code):
```

(continues on next page)

(continued from previous page)

```
    if 'OVC' in code:
        return 1.0
    elif 'BKN' in code:
        return 6.0/8.0
    elif 'SCT' in code:
        return 4.0/8.0
    elif 'FEW' in code:
        return 2.0/8.0
    else:
        return 0

# Pull out these specific stations (prepend K for AWIPS identifiers)
selected = ['PDX', 'OKC', 'ICT', 'GLD', 'MEM', 'BOS', 'MIA', 'MOB', 'ABQ', 'PHX', 'TTF
↪',
            'ORD', 'BIL', 'BIS', 'CPR', 'LAX', 'ATL', 'MSP', 'SLC', 'DFW', 'NYC', 'PHL
↪',
            'PIT', 'IND', 'OLY', 'SYR', 'LEX', 'CHS', 'TLH', 'HOU', 'GJT', 'LBB', 'LSV
↪',
            'GRB', 'CLT', 'LNK', 'DSM', 'BOI', 'FSD', 'RAP', 'RIC', 'JAN', 'HSV', 'CRW
↪',
            'SAT', 'BUY', '0CO', 'ZPC', 'VIH', 'BDG', 'MLF', 'ELY', 'WMC', 'OTH', 'CAR
↪',
            'LMT', 'RDM', 'PDT', 'SEA', 'UIL', 'EPH', 'PUW', 'COE', 'MLP', 'PIH', 'IDA
↪',
            'MSO', 'ACV', 'HLN', 'BIL', 'OLF', 'RUT', 'PSM', 'JAX', 'TPA', 'SHV', 'MSY
↪',
            'ELP', 'RNO', 'FAT', 'SFO', 'NYL', 'BRO', 'MRF', 'DRT', 'FAR', 'BDE', 'DLH
↪',
            'HOT', 'LBF', 'FLG', 'CLE', 'UNV']
selected = ['K{0}'.format(id) for id in selected]
data_arr = []
```

```
# EDEX Request
edexServer = "edex-cloud.unidata.ucar.edu"
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest("obs")
availableProducts = DataAccessLayer.getAvailableParameters(request)

single_value_params = ["timeObs", "stationName", "longitude", "latitude",
                       "temperature", "dewpoint", "windDir",
                       "windSpeed", "seaLevelPress"]
multi_value_params = ["presWeather", "skyCover", "skyLayerBase"]
pres_weather, sky_cov, sky_layer_base = [],[],[]
params = single_value_params + multi_value_params
obs = dict({params: [] for params in params})

request.setParameters(*(params))
request.setLocationNames(*(selected))
```

Here we use the Python-AWIPS class **TimeRange** to prepare a beginning and end time span for requesting observations (the last hour):

```
# Time range
lastHourDateTime = datetime.utcnow() - timedelta(hours = 1)
start = lastHourDateTime.strftime('%Y-%m-%d %H')
beginRange = datetime.strptime( start + ":00:00", "%Y-%m-%d %H:%M:%S")
```

(continues on next page)

(continued from previous page)

```
endRange = datetime.strptime( start + ":59:59", "%Y-%m-%d %H:%M:%S")
timerange = TimeRange(beginRange, endRange)

response = DataAccessLayer.getGeometryData(request,timerange)
```

```
station_names = []
for ob in response:
    avail_params = ob.getParameters()
    if "presWeather" in avail_params:
        pres_weather.append(ob.getString("presWeather"))
    elif "skyCover" in avail_params and "skyLayerBase" in avail_params:
        sky_cov.append(ob.getString("skyCover"))
        sky_layer_base.append(ob.getNumber("skyLayerBase"))
    else:
        # If we already have a record for this stationName, skip
        if ob.getString('stationName') not in station_names:
            station_names.append(ob.getString('stationName'))
            for param in single_value_params:
                if param in avail_params:
                    if param == 'timeObs':
                        obs[param].append(datetime.fromtimestamp(ob.getNumber(param)/
↪1000.0))
                    else:
                        try:
                            obs[param].append(ob.getNumber(param))
                        except TypeError:
                            obs[param].append(ob.getString(param))
                else:
                    obs[param].append(None)

            obs['presWeather'].append(pres_weather);
            obs['skyCover'].append(sky_cov);
            obs['skyLayerBase'].append(sky_layer_base);
            pres_weather = []
            sky_cov = []
            sky_layer_base = []
```

Next grab the simple variables out of the data we have (attaching correct units), and put them into a dictionary that we will hand the plotting function later:

- Get wind components from speed and direction
- Convert cloud fraction values to integer codes [0 - 8]
- Map METAR weather codes to WMO codes for weather symbols

```
data = dict()
data['stid'] = np.array(obs['stationName'])
data['latitude']  = np.array(obs['latitude'])
data['longitude'] = np.array(obs['longitude'])
data['air_temperature'] = np.array(obs['temperature'], dtype=float)* units.degC
data['dew_point_temperature'] = np.array(obs['dewpoint'], dtype=float)* units.degC
data['air_pressure_at_sea_level'] = np.array(obs['seaLevelPress'])* units('mbar')

direction = np.array(obs['windDir'])
direction[direction == -9999.0] = 'nan'
```

(continues on next page)

(continued from previous page)

```
u, v = wind_components(np.array(obs['windSpeed']) * units('knots'),
                       direction * units.degree)
data['eastward_wind'], data['northward_wind'] = u, v
data['cloud_coverage'] = [int(get_cloud_cover(x)*8) for x in obs['skyCover']]
data['present_weather'] = obs['presWeather']
```

```
print(obs['stationName'])
```

```
['K0CO', 'KHOT', 'KSHV', 'KIND', 'KBDE', 'KPSM', 'KORD', 'KDFW', 'KPHL', 'KTTF', 'KBDG
↪', 'KOLY', 'KNYC', 'KABQ', 'KLEX', 'KDRT', 'KELP', 'KRUT', 'KRIC', 'KPIT', 'KMSP',
↪'KHSV', 'KUNV', 'KSAT', 'KCLE', 'KPHX', 'KMIA', 'KBOI', 'KBRO', 'KLAX', 'KLBB',
↪'KMSO', 'KPDX', 'KTLH', 'KUIL', 'KTPA', 'KVIH', 'KBIL', 'KMLF', 'KCPR', 'KATL',
↪'KBIS', 'KCLT', 'KOKC', 'KRAP', 'KACV', 'KEPH', 'KELY', 'KFAR', 'KFAT', 'KMSY',
↪'KOLF', 'KPDT', 'KLMT', 'KHLN', 'KHOU', 'KICT', 'KIDA', 'KPIH', 'KPUW', 'KGJT',
↪'KGLD', 'KGRB', 'KLBF', 'KMLP', 'KBOS', 'KSYR', 'KDLH', 'KCOE', 'KOTH', 'KCRW',
↪'KSEA', 'KCAR', 'KDSM', 'KJAN', 'KSLC', 'KBUY', 'KLNK', 'KMEM', 'KNYL', 'KRDM',
↪'KCHS', 'KFSD', 'KJAX', 'KMOB', 'KRNO', 'KSFO', 'KWMC', 'KFLG', 'KLSV']
```

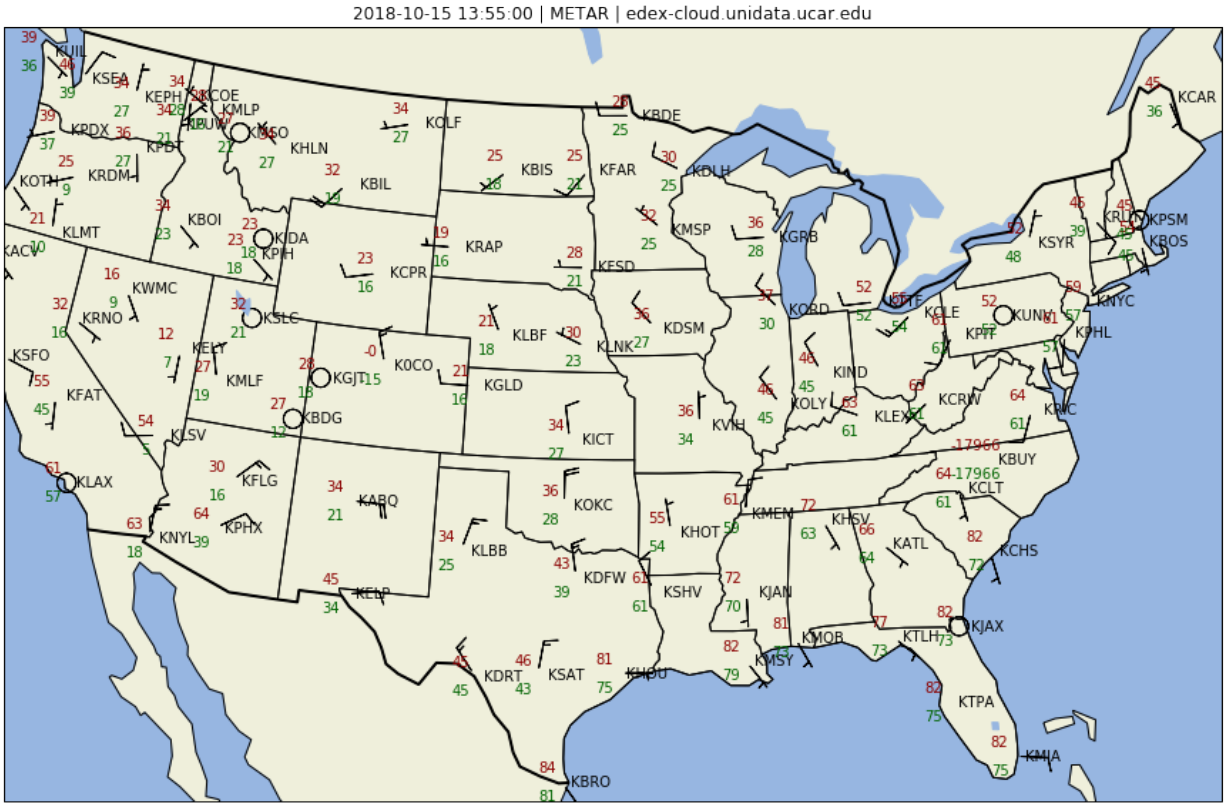
MetPy Surface Obs Plot

```
proj = ccrs.LambertConformal(central_longitude=-95, central_latitude=35,
                             standard_parallels=[35])

# Change the DPI of the figure
plt.rcParams['savefig.dpi'] = 255

# Winds, temps, dewpoint, station id
custom_layout = StationPlotLayout()
custom_layout.add_barb('eastward_wind', 'northward_wind', units='knots')
custom_layout.add_value('NW', 'air_temperature', fmt='.0f', units='degF', color=
↪'darkred')
custom_layout.add_value('SW', 'dew_point_temperature', fmt='.0f', units='degF', color=
↪'darkgreen')
custom_layout.add_value('E', 'precipitation', fmt='0.1f', units='inch', color='blue')

# Create the figure
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(1, 1, 1, projection=proj)

# Add various map elements
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.LAKES)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.STATES)
ax.add_feature(cfeature.BORDERS, linewidth=2)

# Set plot bounds
ax.set_extent((-118, -73, 23, 50))
ax.set_title(str(ob.getDataTime()) + " | METAR | " + edexServer)

stationplot = StationPlot(ax, data['longitude'], data['latitude'], clip_on=True,
                          transform=ccrs.PlateCarree(), fontsize=10)
stationplot.plot_text((2, 0), data['stid'])
```

(continues on next page)

(continued from previous page)

```
custom_layout.plot(stationplot, data)
plt.show()
```



Map Resources and Topography

Notebook

The python-awips package provides access to the entire AWIPS Maps Database for use in Python GIS applications. Map objects are returned as Shapely geometries (*Polygon*, *Point*, *MultiLineString*, etc.) and can be easily plotted by Matplotlib, Cartopy, MetPy, and other packages.

Each map database table has a geometry field called `the_geom`, which can be used to spatially select map resources for any column of type geometry,

Notes

- This notebook requires: **python-awips, numpy, matplotlib, cartopy, shapely**
- Use datatype **maps** and **addIdentifier('table', <postgres maps schema>)** to define the map table: DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu") request = DataAccessLayer.newDataRequest('maps') request.addIdentifier('table', 'mapdata.county')
- Use **request.setLocationNames()** and **request.addIdentifier()** to spatially filter a map resource. In the example below, WFO ID **BOU** (Boulder, Colorado) is used to query counties within the BOU county watch area (CWA)

```
request.addIdentifier('geomField', 'the_geom')
request.addIdentifier('inLocation', 'true')
request.addIdentifier('locationField', 'cwa')
request.setLocationNames('BOU')
request.addIdentifier('cwa', 'BOU')
```

See the Maps Database Reference Page for available database tables, column names, and types.

> Note the geometry definition of `the_geom` for each data type, which can be **Point**, **MultiPolygon**, or **MultiLineString**.

Setup

```
from __future__ import print_function
from awips.dataaccess import DataAccessLayer
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import numpy as np
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
from cartopy.feature import ShapelyFeature,NaturalEarthFeature
from shapely.geometry import Polygon
from shapely.ops import cascaded_union

# Standard map plot
def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(12,12),
            subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax

# Server, Data Request Type, and Database Table
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest('maps')
request.addIdentifier('table', 'mapdata.county')
```

Request County Boundaries for a WFO

- Use **request.setParameters()** to define fields to be returned by the request.

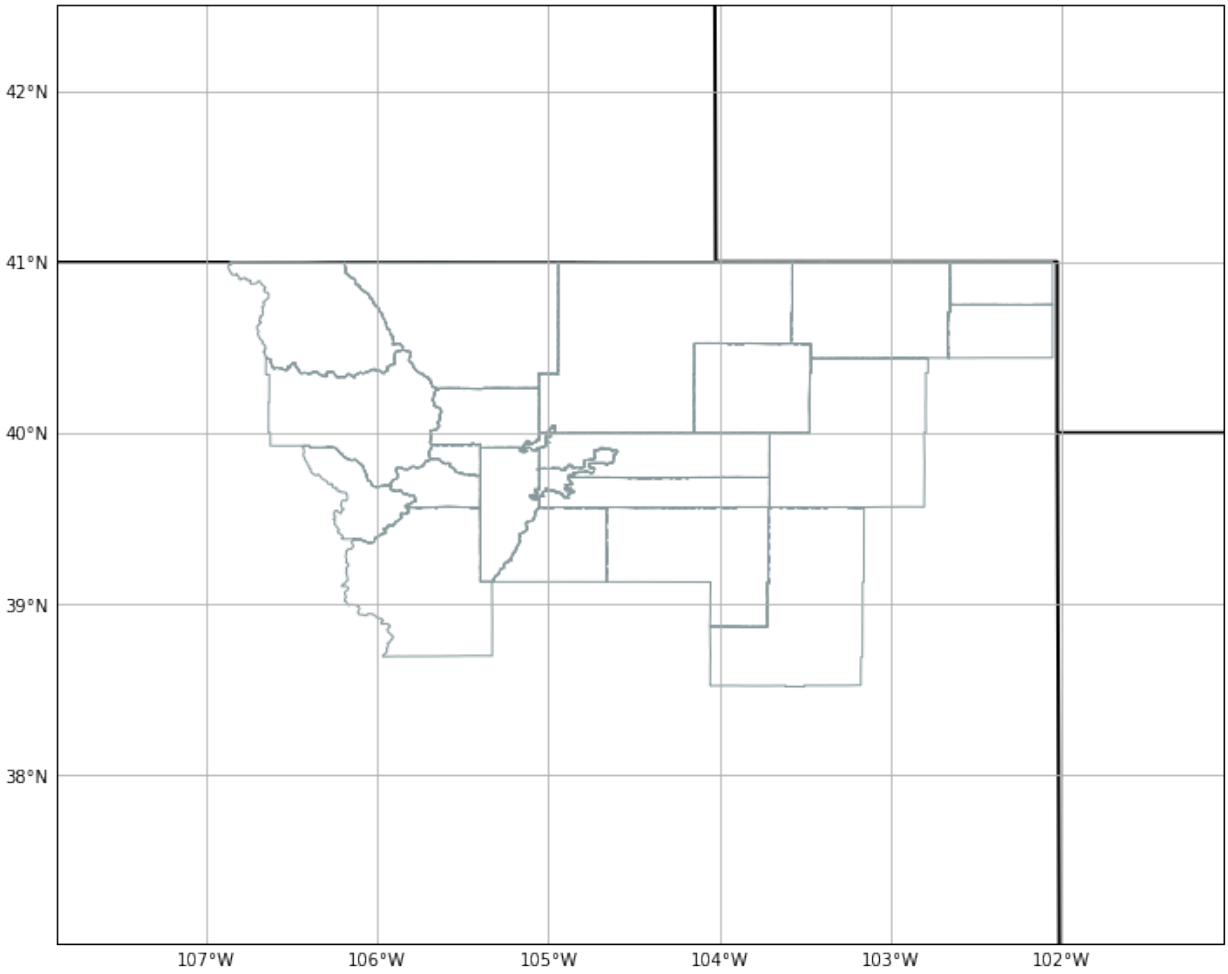
```
# Define a WFO ID for location
# tie this ID to the mapdata.county column "cwa" for filtering
request.setLocationNames('BOU')
request.addIdentifier('cwa', 'BOU')

# enable location filtering (inLocation)
# locationField is tied to the above cwa definition (BOU)
request.addIdentifier('geomField', 'the_geom')
request.addIdentifier('inLocation', 'true')
request.addIdentifier('locationField', 'cwa')
```

(continues on next page)

(continued from previous page)

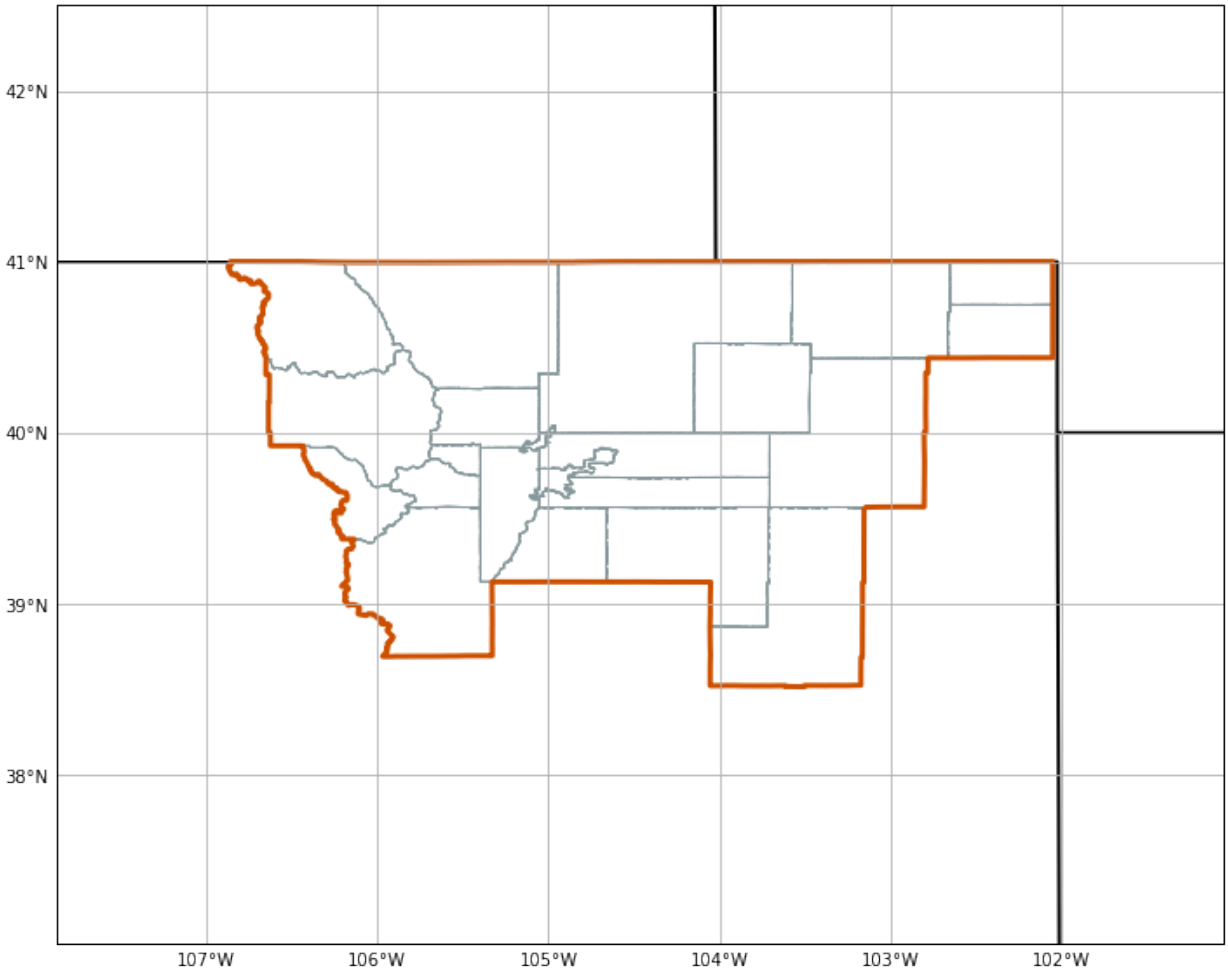
```
# This is essentially the same as ""select count(*) from mapdata.cwa where cwa='BOU';
↪" (=1)

# Get response and create dict of county geometries
response = DataAccessLayer.getGeometryData(request, [])
counties = np.array([])
for ob in response:
    counties = np.append(counties,ob.getGeometry())
print("Using " + str(len(counties)) + " county MultiPolygons")


%matplotlib inline
# All WFO counties merged to a single Polygon
merged_counties = cascaded_union(counties)
envelope = merged_counties.buffer(2)
boundaries=[merged_counties]

# Get bounds of this merged Polygon to use as buffered map extent
bounds = merged_counties.bounds
bbox=[bounds[0]-1,bounds[2]+1,bounds[1]-1.5,bounds[3]+1.5]


fig, ax = make_map(bbox=bbox)
# Plot political/state boundaries handled by Cartopy
political_boundaries = NaturalEarthFeature(category='cultural',
                                name='admin_0_boundary_lines_land',
                                scale='50m', facecolor='none')
states = NaturalEarthFeature(category='cultural',
                                name='admin_1_states_provinces_lines',
                                scale='50m', facecolor='none')
ax.add_feature(political_boundaries, linestyle='-', edgecolor='black')
ax.add_feature(states, linestyle='-', edgecolor='black',linewidth=2)

# Plot CWA counties
for i, geom in enumerate(counties):
    cbounds = Polygon(geom)
    intersection = cbounds.intersection
    geoms = (intersection(geom)
         for geom in counties
         if cbounds.intersects(geom))
    shape_feature = ShapelyFeature(geoms,ccrs.PlateCarree(),
                        facecolor='none', linestyle="-",edgecolor='#86989B')
    ax.add_feature(shape_feature)
```

```
Using 23 county MultiPolygons
```



Create a merged CWA with cascaded_union

```
# Plot CWA envelope
for i, geom in enumerate(boundaries):
    gbounds = Polygon(geom)
    intersection = gbounds.intersection
    geoms = (intersection(geom)
         for geom in boundaries
         if gbounds.intersects(geom))
    shape_feature = ShapelyFeature(geoms,ccrs.PlateCarree(),
                        facecolor='none', linestyle="-",linewidth=3.,edgecolor='
↪#cc5000')
    ax.add_feature(shape_feature)

fig
```

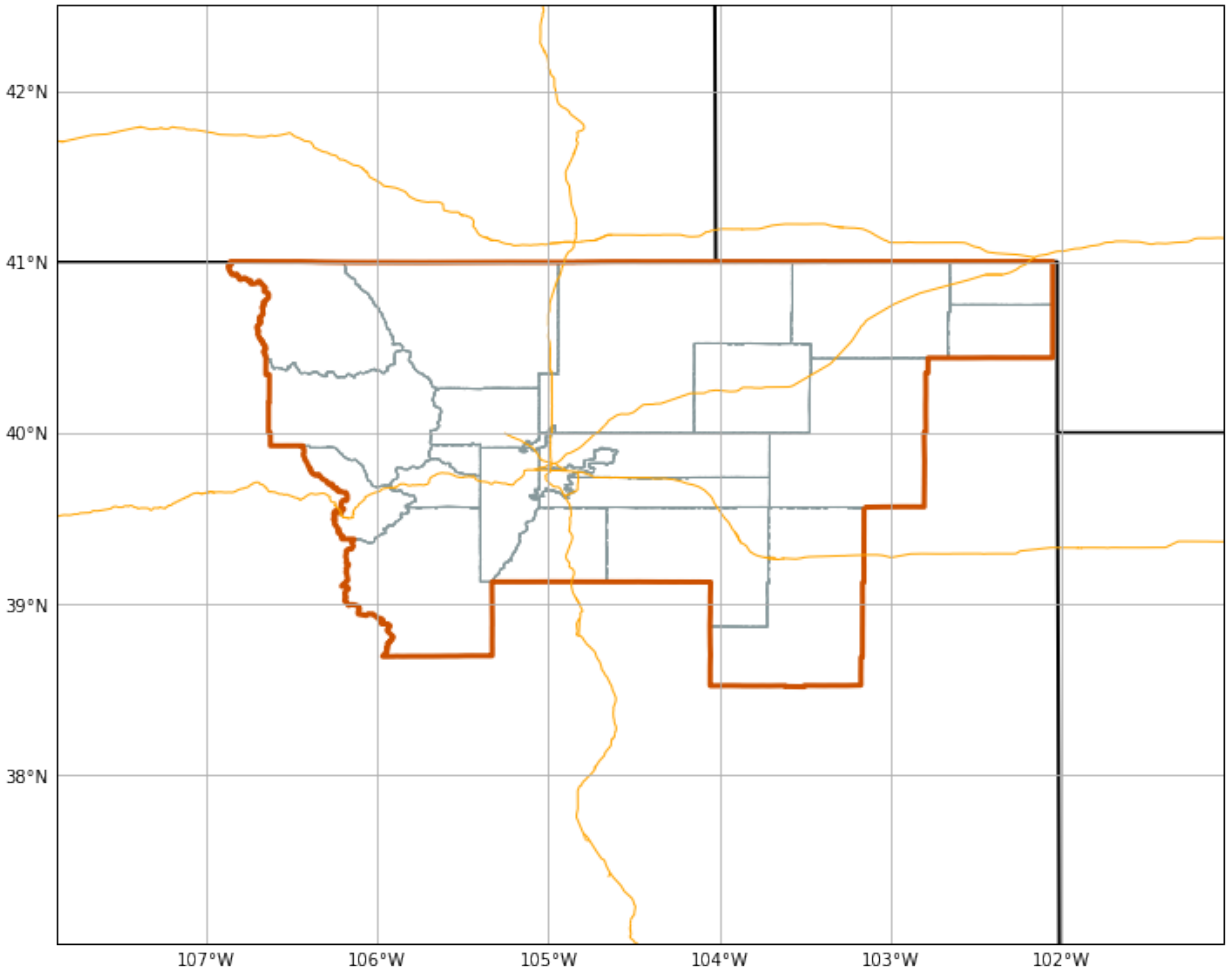


WFO boundary spatial filter for interstates

Using the previously-defined **envelope=merged_counties.buffer(2)** in **newDataRequest()** to request geometries which fall inside the buffered boundary.

```
request = DataAccessLayer.newDataRequest('maps', envelope=envelope)
request.addIdentifier('table', 'mapdata.interstate')
request.addIdentifier('geomField', 'the_geom')
request.setParameters('name')
interstates = DataAccessLayer.getGeometryData(request, [])
print("Using " + str(len(interstates)) + " interstate MultiLineStrings")

# Plot interstates
for ob in interstates:
    shape_feature = ShapelyFeature(ob.getGeometry(),ccrs.PlateCarree(),
                        facecolor='none', linestyle="-",edgecolor='orange')
    ax.add_feature(shape_feature)
fig
```

```
Using 225 interstate MultiLineStrings
```



Nearby cities

Request the city table and filter by population and progressive disclosure level:

Warning: the `prog_disc` field is not entirely understood and values appear to change significantly depending on WFO site.

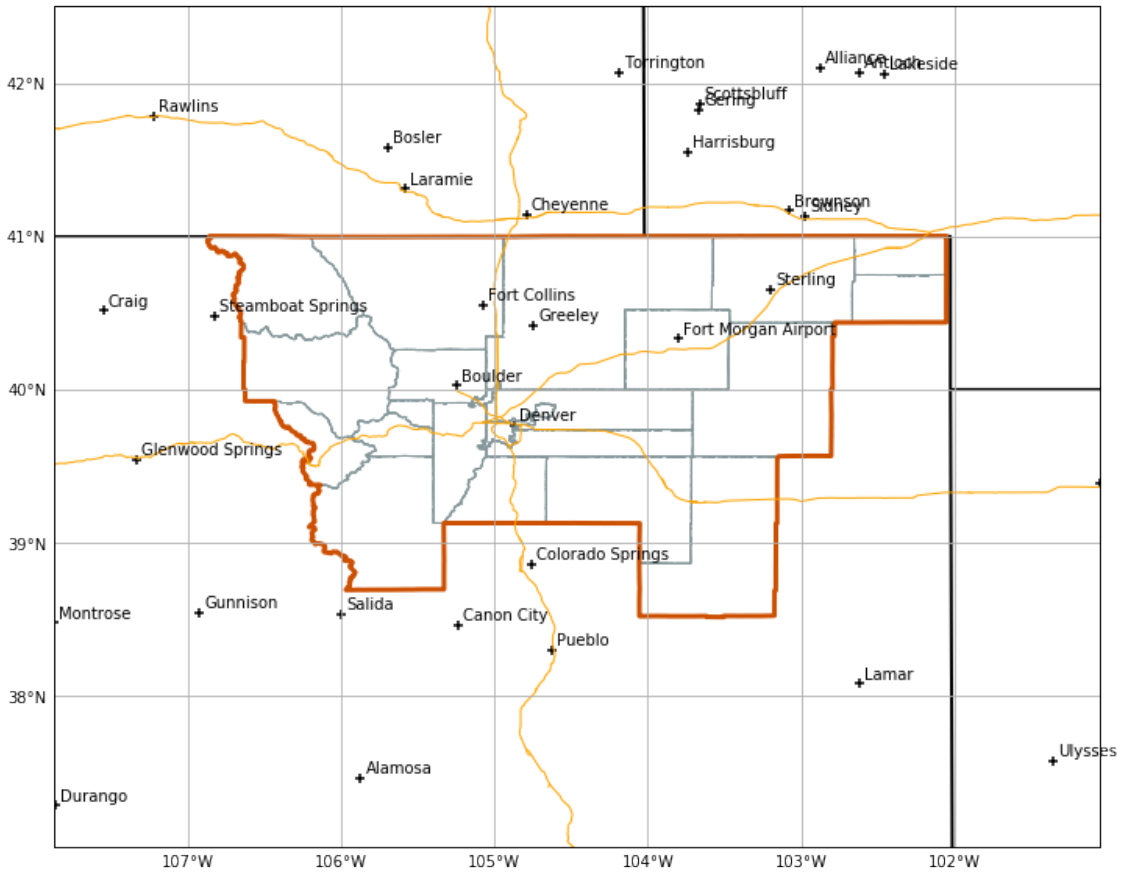
```
request = DataAccessLayer.newDataRequest('maps', envelope=envelope)
request.addIdentifier('table', 'mapdata.city')
request.addIdentifier('geomField', 'the_geom')
request.setParameters('name','population','prog_disc')
cities = DataAccessLayer.getGeometryData(request, [])
print("Queried " + str(len(cities)) + " total cities")

citylist = []
cityname = []
# For BOU, progressive disclosure values above 50 and pop above 5000 looks good
for ob in cities:
    if ob.getString("population"):
        if ob.getNumber("prog_disc") > 50:
            if int(ob.getString("population")) > 5000:
                citylist.append(ob.getGeometry())
```

(continues on next page)

(continued from previous page)

```
                cityname.append(ob.getString("name"))
print("Plotting " + str(len(cityname)) + " cities")

# Plot city markers
ax.scatter([point.x for point in citylist],
       [point.y for point in citylist],
       transform=ccrs.Geodetic(),marker="+",facecolor='black')
# Plot city names
for i, txt in enumerate(cityname):
    ax.annotate(txt, (citylist[i].x,citylist[i].y),
                xytext=(3,3), textcoords="offset points")

fig
```

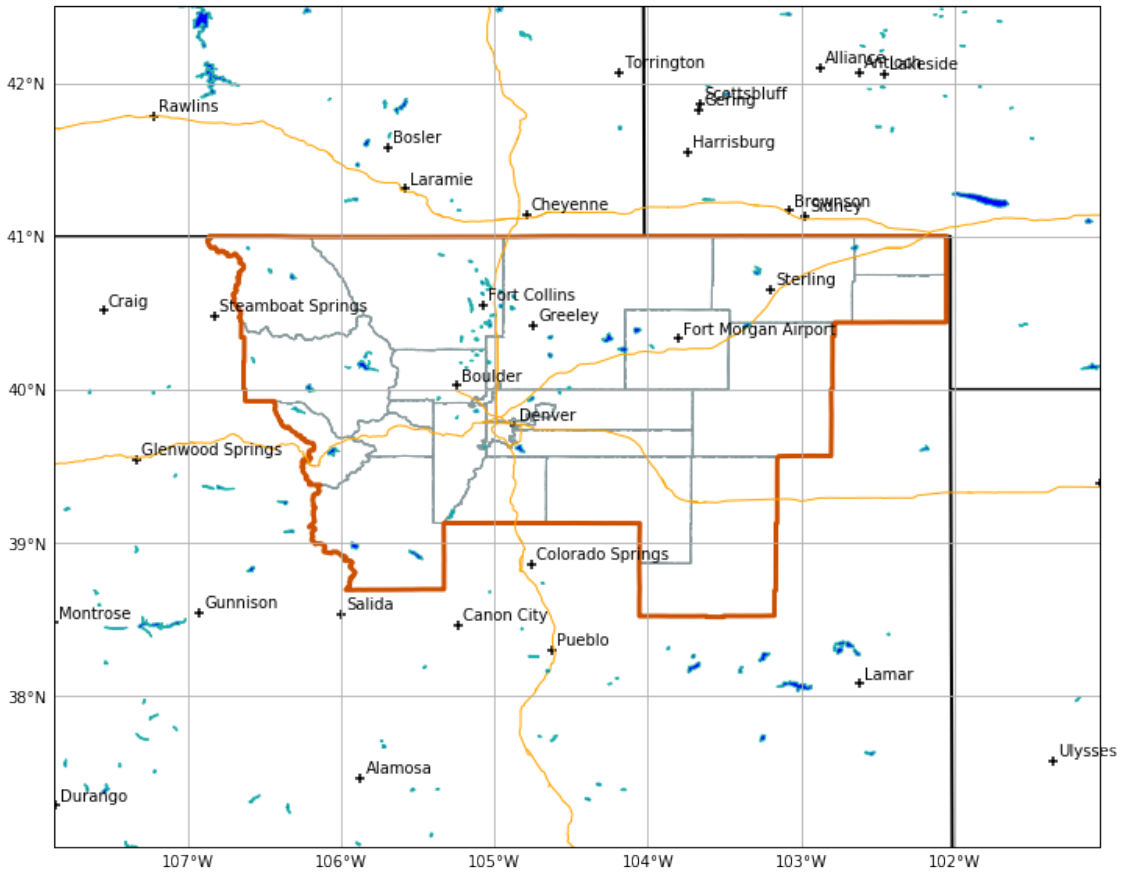
```
Queried 1205 total cities
Plotting 57 cities
```



Lakes

```
request = DataAccessLayer.newDataRequest('maps', envelope=envelope)
request.addIdentifier('table', 'mapdata.lake')
request.addIdentifier('geomField', 'the_geom')
request.setParameters('name')

# Get lake geometries
response = DataAccessLayer.getGeometryData(request, [])
lakes = np.array([])
for ob in response:
    lakes = np.append(lakes,ob.getGeometry())
print("Using " + str(len(lakes)) + " lake MultiPolygons")

# Plot lakes
for i, geom in enumerate(lakes):
    cbounds = Polygon(geom)
    intersection = cbounds.intersection
    geoms = (intersection(geom)
         for geom in lakes
         if cbounds.intersects(geom))
    shape_feature = ShapelyFeature(geoms,ccrs.PlateCarree(),
                        facecolor='blue', linestyle="-",edgecolor='#20B2AA')
    ax.add_feature(shape_feature)
fig
```

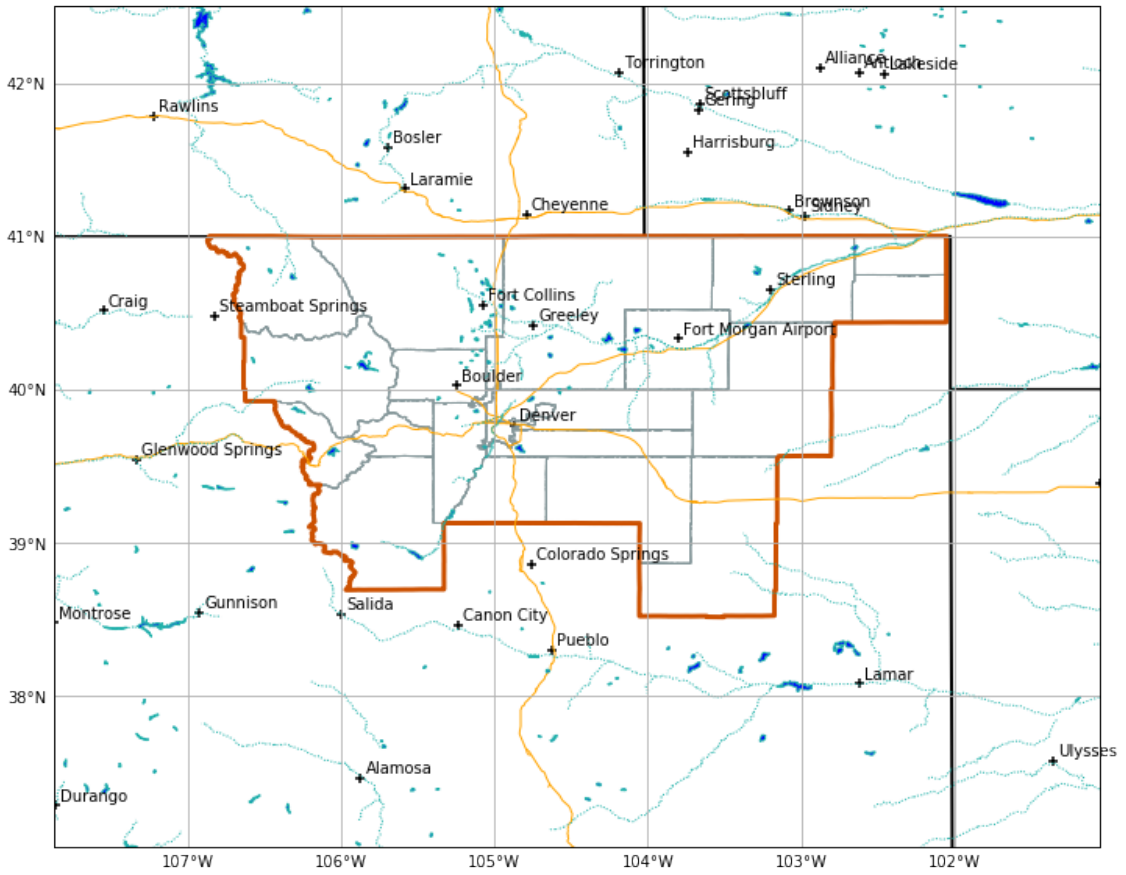
```
Using 208 lake MultiPolygons
```



Major Rivers

```
request = DataAccessLayer.newDataRequest('maps', envelope=envelope)
request.addIdentifier('table', 'mapdata.majorrivers')
request.addIdentifier('geomField', 'the_geom')
request.setParameters('pname')
rivers = DataAccessLayer.getGeometryData(request, [])
print("Using " + str(len(rivers)) + " river MultiLineStrings")

# Plot rivers
for ob in rivers:
    shape_feature = ShapelyFeature(ob.getGeometry(),ccrs.PlateCarree(),
                        facecolor='none', linestyle=":",edgecolor='#20B2AA')
    ax.add_feature(shape_feature)
fig
```

```
Using 1400 river MultiLineStrings
```



Topography

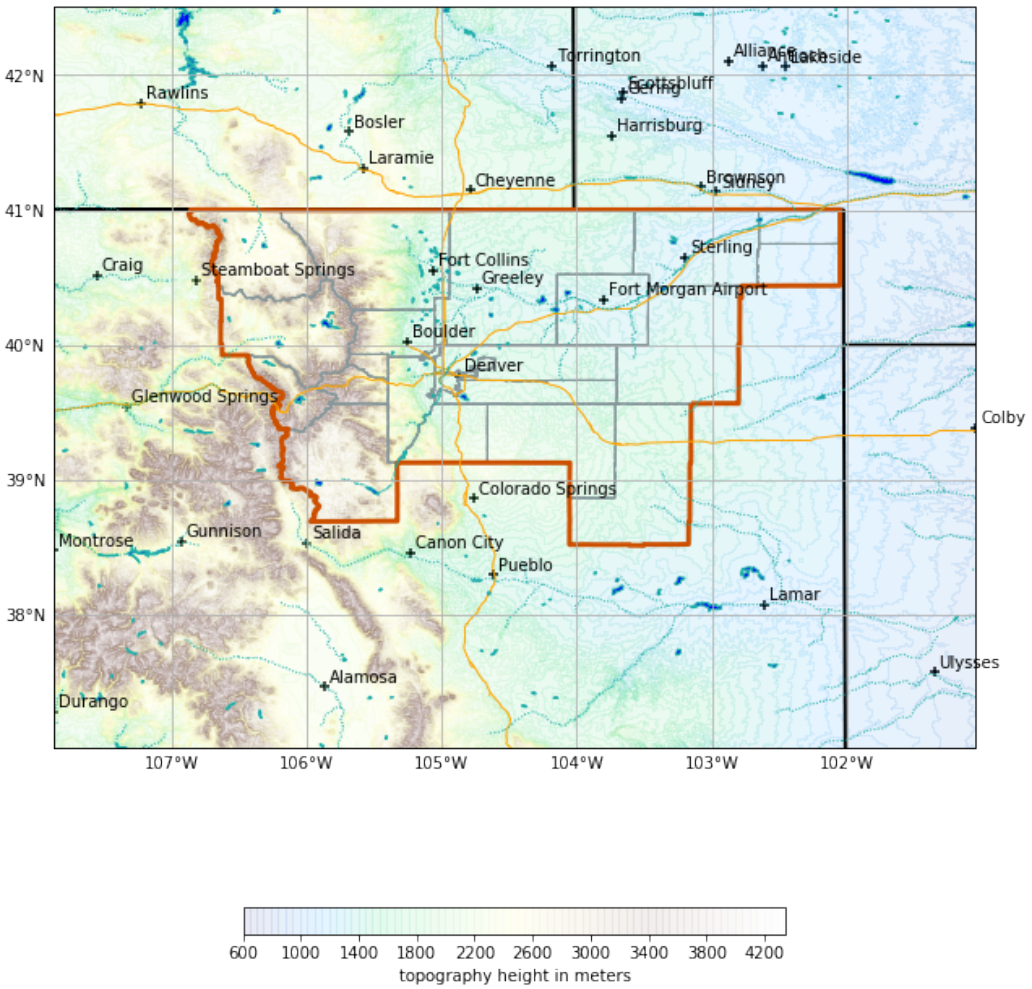
Spatial envelopes are required for topo requests, which can become slow to download and render for large (CONUS) maps.

```
import numpy.ma as ma
request = DataAccessLayer.newDataRequest("topo")
request.addIdentifier("group", "/")
request.addIdentifier("dataset", "full")
request.setEnvelope(envelope)
gridData = DataAccessLayer.getGridData(request)
print(gridData)
print("Number of grid records: " + str(len(gridData)))
print("Sample grid data shape:\n" + str(gridData[0].getRawData().shape) + "\n")
print("Sample grid data:\n" + str(gridData[0].getRawData()) + "\n")
```

```
[<awips.dataaccess.PyGridData.PyGridData object at 0x1165d15f8>]
Number of grid records: 1
Sample grid data shape:
(778, 1058)

Sample grid data:
[[1694. 1693. 1688. ...  757.  761.  762.]
 [1701. 1701. 1701. ...  758.  760.  762.]
 [1703. 1703. 1703. ...  760.  761.  762.]
 ...
 [1767. 1741. 1706. ...  769.  762.  768.]
 [1767. 1746. 1716. ...  775.  765.  761.]
 [1781. 1753. 1730. ...  766.  762.  759.]]
```

```
grid=gridData[0]
topo=ma.masked_invalid(grid.getRawData())
lons, lats = grid.getLatLonCoords()
print(topo.min()) # minimum elevation in our domain (meters)
print(topo.max()) # maximum elevation in our domain (meters)

# Plot topography
cs = ax.contourf(lons, lats, topo, 80, cmap=plt.get_cmap('terrain'),alpha=0.1)
cbar = fig.colorbar(cs, extend='both', shrink=0.5, orientation='horizontal')
cbar.set_label("topography height in meters")

fig
```

```
623.0
4328.0
```



Model Sounding Data

Notebook

The EDEX modelsounding plugin creates 64-level vertical profiles from GFS and ETA (NAM) BUFR products distirubted over NOAAport. Paramters which are requestable are **pressure**, **temperature**, **specHum**, **uComp**, **vComp**, **omega**, **cldCvr**.

```
from awips.dataaccess import DataAccessLayer
import matplotlib.tri as mtri
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
from math import exp, log
```

(continues on next page)

(continued from previous page)

```
import numpy as np
from metpy.calc import get_wind_components, lcl, dry_lapse, parcel_profile, dewpoint
from metpy.calc import wind_speed, wind_direction, thermo, vapor_pressure
from metpy.plots import SkewT, Hodograph
from metpy.units import units, concatenate
import warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)

DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest("modelsounding")
forecastModel = "GFS"
request.addIdentifier("reportType", forecastModel)
request.setParameters("pressure","temperature","specHum","uComp","vComp","omega",
↪"cldCvr")
```

Available Locations

```
locations = DataAccessLayer.getAvailableLocationNames(request)
locations.sort()
list(locations)
```

```
['',
 '1V4',
 '3J2',
 '4BL',
 '4BQ',
 '4HV',
 '4OM',
 '5AF',
 '5AG',
 '5SZ',
 '6RO',
 '8V7',
 '9B6',
 'A#2',
 'A#3',
 'A#4',
 'A#5',
 'A#6',
 'A#7',
 'A#8',
 'A#9',
 'A#A',
 'A#B',
 'ABL',
 'ADM',
 'AFA',
 'AGR',
 'AHN',
 'AIA',
 'AIH',
 'AJO',
 'ANJ',
 'APX',
```

(continues on next page)

(continued from previous page)

```
'AQQ',
'ATH',
'ATL1',
'ATL2',
'ATL3',
'ATL4',
'ATLH',
'AWH',
'AWR',
'B#1',
'B#2',
'B#3',
'B#4',
'B#5',
'B#6',
'B#7',
'B#8',
'B#9',
'B#A',
'B#B',
'B#C',
'B#D',
'B#E',
'B#F',
'B#G',
'B#H',
'B#J',
'B#K',
'B#L',
'B#M',
'B#N',
'B#O',
'B#P',
'B#Q',
'B#S',
'BAB',
'BDG',
'BDP',
'BFL',
'BGTL',
'BH1',
'BH2',
'BH3',
'BH4',
'BH5',
'BHK',
'BID',
'BIR',
'BLS',
'BLU',
'BMX',
'BNA',
'BOD',
'BRA',
'BTL',
'BVR',
'C01',
```

(continues on next page)

(continued from previous page)

```
'C02',
'C03',
'C04',
'C06',
'C07',
'C08',
'C09',
'C10',
'C11',
'C12',
'C13',
'C14',
'C17',
'C18',
'C19',
'C20',
'C21',
'C22',
'C23',
'C24',
'C25',
'C27',
'C28',
'C30',
'C31',
'C32',
'C33',
'C34',
'C35',
'C36',
'C7H',
'CAI',
'CAN',
'CBE',
'CBN',
'CHE',
'CKN',
'CLD',
'CLE',
'CLN',
'COL1',
'COL2',
'COL3',
'COL4',
'COT',
'CQV',
'CRL',
'CRR',
'CTY',
'CVM',
'CVS',
'CWEU',
'CWFN',
'CWKX',
'CWLB',
'CWLO',
'CWLT',
```

(continues on next page)

(continued from previous page)

```
'CWLW',
'CWMW',
'CWOS',
'CWPH',
'CWQG',
'CWSA',
'CWSE',
'CWZB',
'CWZC',
'CWZV',
'CYAH',
'CYAW',
'CYBK',
'CYBU',
'CYCB',
'CYCG',
'CYCX',
'CYDA',
'CYEG',
'CYEV',
'CYFB',
'CYFO',
'CYFS',
'CYGQ',
'CYHM',
'CYHZ',
'CYJT',
'CYLH',
'CYLJ',
'CYMD',
'CYMO',
'CYMT',
'CYMX',
'CYOC',
'CYOW',
'CYPA',
'CYPE',
'CYPL',
'CYPQ',
'CYQA',
'CYQD',
'CYQG',
'CYQH',
'CYQI',
'CYQK',
'CYQQ',
'CYQR',
'CYQT',
'CYQX',
'CYQY',
'CYRB',
'CYSM',
'CYSY',
'CYTH',
'CYTL',
'CYTS',
'CYUL',
```

(continues on next page)

(continued from previous page)

```
'CYUX',
'CYVO',
'CYVP',
'CYVQ',
'CYVR',
'CYVV',
'CYWA',
'CYWG',
'CYWO',
'CYXC',
'CYXE',
'CYXH',
'CYXS',
'CYXU',
'CYXX',
'CYXY',
'CYXZ',
'CYYB',
'CYYC',
'CYYE',
'CYYJ',
'CYYQ',
'CYYR',
'CYYT',
'CYYZ',
'CYZF',
'CYZS',
'CYZT',
'CYZV',
'DEN',
'DOV',
'DPG',
'DSC',
'DSD',
'DTX',
'DVN',
'DYS',
'E28',
'E74',
'EAT',
'EAX',
'EDW',
'EFL',
'EMP',
'END',
'ENL',
'ESTC',
'FCS',
'FDR',
'FFC',
'FHU',
'FLG',
'FLP',
'FPK',
'FRI',
'FSI',
'FTR',
```

(continues on next page)

(continued from previous page)

```
'FWD',
'G#1',
'G#2',
'G#3',
'G#4',
'G#5',
'G#6',
'G#7',
'G#8',
'G#9',
'G#A',
'G#B',
'G#C',
'G#D',
'G#E',
'G#F',
'G#G',
'G001',
'G003',
'G004',
'G005',
'G007',
'G009',
'GDP',
'GDV',
'GLRY',
'GMX1',
'GNB',
'GNC',
'GRF',
'GTB',
'GTP',
'GVL',
'GVS',
'GYX',
'H02',
'HAY',
'HGR',
'HMN',
'HOM',
'HOO',
'HSI',
'HYR',
'HYS',
'ICC',
'IGM',
'ILN',
'ILS',
'ILX',
'IMT',
'INK',
'IPX',
'JACK',
'JDN',
'K40B',
'K9V9',
'KABE',
```

(continues on next page)

(continued from previous page)

```
 'KABI',
 'KABQ',
 'KABR',
 'KABY',
 'KACK',
 'KACT',
 'KACV',
 'KACY',
 'KAGC',
 'KAGS',
 'KAHN',
 'KAK',
 'KALB',
 'KALI',
 'KALO',
 'KALS',
 'KALW',
 'KAMA',
 'KAN',
 'KANB',
 'KAND',
 'KAOO',
 'KAPA',
 'KAPN',
 'KART',
 'KASE',
 'KAST',
 'KATL',
 'KATY',
 'KAUG',
 'KAUS',
 'KAUW',
 'KAVL',
 'KAVP',
 'KAXN',
 'KAYS',
 'KAZO',
 'KBAF',
 'KBCE',
 'KBDE',
 'KBDL',
 'KBDR',
 'KBED',
 'KBFD',
 'KBFF',
 'KBFI',
 'KBFL',
 'KBGM',
 'KBGR',
 'KBHB',
 'KBHM',
 'KBIH',
 'KBIL',
 'KBIS',
 'KBJC',
 'KBJI',
 'KBKE',
```

(continues on next page)

(continued from previous page)

```
'KBKW',
'KBLF',
'KBLH',
'KBLI',
'KBML',
'KBNA',
'KBNO',
'KBNV',
'KBOI',
'KBOS',
'KBPT',
'KBQK',
'KBRD',
'KBRL',
'KBRO',
'KBTL',
'KBTM',
'KBTR',
'KBTV',
'KBUF',
'KBUR',
'KBVI',
'KBVX',
'KBVY',
'KBWG',
'KBWI',
'KBYI',
'KBZN',
'KCAE',
'KCAK',
'KCAR',
'KCDC',
'KCDR',
'KCDS',
'KCEC',
'KCEF',
'KCGI',
'KCGX',
'KCHA',
'KCHH',
'KCHO',
'KCHS',
'KCID',
'KCIU',
'KCKB',
'KCKL',
'KCLE',
'KCLL',
'KCLM',
'KCLT',
'KCMH',
'KCMI',
'KCMX',
'KCNM',
'KCNU',
'KCOD',
'KCOE',
```

(continues on next page)

(continued from previous page)

```
'KCON',
'KCOS',
'KCOU',
'KCPR',
'KCRE',
'KCRP',
'KCRQ',
'KCRW',
'KCSG',
'KCSV',
'KCTB',
'KCVG',
'KCWA',
'KCYS',
'KDAB',
'KDAG',
'KDAL',
'KDAN',
'KDAY',
'KDBQ',
'KDCA',
'KDDC',
'KDEC',
'KDEN',
'KDET',
'KDFW',
'KDHN',
'KDHT',
'KDIK',
'KDLH',
'KDLS',
'KDMN',
'KDPA',
'KDRA',
'KDRO',
'KDRT',
'KDSM',
'KDTW',
'KDUG',
'KDUJ',
'KEAT',
'KEAU',
'KECG',
'KEED',
'KEGE',
'KEKN',
'KEKO',
'KEL',
'KELD',
'KELM',
'KELO',
'KELP',
'KELY',
'KENV',
'KEPH',
'KEPO',
'KEPZ',
```

(continues on next page)

(continued from previous page)

```
'KERI',
'KESF',
'KEUG',
'KEVV',
'KEWB',
'KEWN',
'KEWR',
'KEYW',
'KFAM',
'KFAR',
'KFAT',
'KFAY',
'KFCA',
'KFDY',
'KFKL',
'KFLG',
'KFLL',
'KFLO',
'KFMN',
'KFMY',
'KFNT',
'KFOE',
'KFPR',
'KFRM',
'KFSD',
'KFSM',
'KFTW',
'KFTY',
'KFVE',
'KFVX',
'KFWA',
'KFXE',
'KFYV',
'KGAG',
'KGCC',
'KGCK',
'KGCN',
'KGEG',
'KGFK',
'KGFL',
'KGGG',
'KGGW',
'KGJT',
'KGLD',
'KGLH',
'KGLS',
'KGMU',
'KGNR',
'KGNV',
'KGON',
'KGPT',
'KGRB',
'KGRI',
'KGRR',
'KGSO',
'KGSP',
'KGTF',
```

(continues on next page)

(continued from previous page)

```
'KGUC',
'KGUP',
'KGWO',
'KGYY',
'KGZH',
'KHAT',
'KHBR',
'KHDN',
'KHIB',
'KHIO',
'KHKY',
'KHLG',
'KHLN',
'KHOB',
'KHON',
'KHOT',
'KHOU',
'KHPN',
'KHQM',
'KHRL',
'KHRO',
'KHSV',
'KHTH',
'KHTS',
'KHUF',
'KHUL',
'KHUT',
'KHVN',
'KHVR',
'KHYA',
'KIAD',
'KIAG',
'KIAH',
'KICT',
'KIDA',
'KIL',
'KILG',
'KILM',
'KIND',
'KINK',
'KINL',
'KINT',
'KINW',
'KIPL',
'KIPT',
'KISN',
'KISP',
'KITH',
'KIWD',
'KJAC',
'KJAN',
'KJAX',
'KJBR',
'KJFK',
'KJHW',
'KJKL',
'KJLN',
```

(continues on next page)

(continued from previous page)

```
'KJMS',
'KJST',
'KJXN',
'KKL',
'KLAF',
'KLAN',
'KLAR',
'KLAS',
'KLAX',
'KLBB',
'KLBE',
'KLBF',
'KLCB',
'KLCH',
'KLEB',
'KLEX',
'KLFK',
'KLFT',
'KLGA',
'KLGB',
'KLGU',
'KLIT',
'KLMT',
'KLND',
'KLNK',
'KLOL',
'KLOZ',
'KLRD',
'KLSE',
'KLUK',
'KLVS',
'KLWB',
'KLWM',
'KLWS',
'KLWT',
'KLYH',
'KLZK',
'KMAF',
'KMBS',
'KMCB',
'KMCE',
'KMCI',
'KMCN',
'KMCO',
'KMCW',
'KMDN',
'KMDT',
'KMDW',
'KMEI',
'KMEM',
'KMFD',
'KMFE',
'KMFR',
'KMGM',
'KMGW',
'KMHE',
'KMHK',
```

(continues on next page)

(continued from previous page)

```
'KMHT',
'KMHX',
'KMIA',
'KMIV',
'KMKC',
'KMKE',
'KMKG',
'KMKL',
'KMLB',
'KMLC',
'KMLI',
'KMLS',
'KMLT',
'KMLU',
'KMMU',
'KMOB',
'KMOT',
'KMPV',
'KMQT',
'KMRB',
'KMRY',
'KMSL',
'KMSN',
'KMSO',
'KMSP',
'KMSS',
'KMSY',
'KMTJ',
'KMTN',
'KMWH',
'KMYR',
'KNA',
'KNEW',
'KNL',
'KNSI',
'KOAK',
'KOFK',
'KOGD',
'KOKC',
'KOLM',
'KOMA',
'KONT',
'KOPF',
'KOQU',
'KORD',
'KORF',
'KORH',
'KOSH',
'KOTH',
'KOTM',
'KP11',
'KP38',
'KPAE',
'KPAH',
'KPBF',
'KPBI',
'KPDK',
```

(continues on next page)

(continued from previous page)

```
'KPDT',
'KPDX',
'KPFN',
'KPGA',
'KPHF',
'KPHL',
'KPHN',
'KPHX',
'KPIA',
'KPIB',
'KPIE',
'KPIH',
'KPIR',
'KPIT',
'KPKB',
'KPLN',
'KPMD',
'KPNC',
'KPNE',
'KPNS',
'KPOU',
'KPQI',
'KPRB',
'KPRC',
'KPSC',
'KPSM',
'KPSP',
'KPTK',
'KPUB',
'KPVD',
'KPVU',
'KPWM',
'KRAD',
'KRAP',
'KRBL',
'KRDD',
'KRDG',
'KRDM',
'KRDU',
'KRFD',
'KRIC',
'KRIW',
'KRKD',
'KRKS',
'KRNO',
'KRNT',
'KROA',
'KROC',
'KROW',
'KRSL',
'KRST',
'KRSW',
'KRUM',
'KRWF',
'KRWI',
'KRWL',
'KSAC',
```

(continues on next page)

(continued from previous page)

```
'KSAF',
'KSAN',
'KSAT',
'KSAV',
'KSBA',
'KSBN',
'KSBP',
'KSBY',
'KSCH',
'KSCK',
'KSDF',
'KSDM',
'KSDY',
'KSEA',
'KSEP',
'KSFF',
'KSFO',
'KSGF',
'KSGU',
'KSHR',
'KSHV',
'KSJC',
'KSJT',
'KSLC',
'KSLE',
'KSLK',
'KSLN',
'KSMF',
'KSMX',
'KSNA',
'KSNS',
'KSPI',
'KSPS',
'KSRQ',
'KSSI',
'KSTJ',
'KSTL',
'KSTP',
'KSTS',
'KSUN',
'KSUS',
'KSUX',
'KSVE',
'KSWF',
'KSYR',
'KTCC',
'KTCL',
'KTCS',
'KTEB',
'KTIW',
'KTLH',
'KTMB',
'KTOL',
'KTOP',
'KTPA',
'KTPH',
'KTRI',
```

(continues on next page)

(continued from previous page)

```
'KTRK',
'KTRM',
'KTTD',
'KTTN',
'KTUL',
'KTUP',
'KTUS',
'KTVC',
'KTVL',
'KTWF',
'KTXK',
'KTYR',
'KTYS',
'KUCA',
'KUIN',
'KUKI',
'KUNV',
'KVCT',
'KVEL',
'KVLD',
'KVNY',
'KVRB',
'KWJF',
'KWMC',
'KWRL',
'KWYS',
'KY22',
'KY26',
'KYKM',
'KYKN',
'KYNG',
'KYUM',
'KZZV',
'LAA',
'LAP',
'LBY',
'LDL',
'LHX',
'LIC',
'LOR',
'LRR',
'LSF',
'LUS',
'LVM',
'LW1',
'MAC',
'MAX',
'MAZ',
'MDPC',
'MDPP',
'MDSD',
'MDST',
'MGFL',
'MGGT',
'MGHT',
'MGPB',
'MGSJ',
```

(continues on next page)

(continued from previous page)

```
'MHAM',
'MHCA',
'MHCH',
'MHLC',
'MHLE',
'MHLM',
'MHNJ',
'MHPL',
'MHRO',
'MHSR',
'MHTE',
'MHTG',
'MHYR',
'MIB',
'MIE',
'MKJP',
'MKJS',
'MLD',
'MMAA',
'MMAS',
'MMBT',
'MMCE',
'MMCL',
'MMCN',
'MMCS',
'MMCU',
'MMCV',
'MMCZ',
'MMDO',
'MMGL',
'MMGM',
'MMHO',
'MMLP',
'MMMA',
'MMMD',
'MMML',
'MMMM',
'MMMT',
'MMMX',
'MMMY',
'MMMZ',
'MMNL',
'MMPR',
'MMRX',
'MMSD',
'MMSP',
'MMTC',
'MMTJ',
'MMTM',
'MMTO',
'MMTP',
'MMUN',
'MMVR',
'MMZC',
'MMZH',
'MMZO',
'MNMG',
```

(continues on next page)

(continued from previous page)

```
 'MNPC',
 'MOR',
 'MPBO',
 'MPCH',
 'MPDA',
 'MPMG',
 'MPSA',
 'MPTO',
 'MPX',
 'MRCH',
 'MRF',
 'MRLB',
 'MRLM',
 'MROC',
 'MRPV',
 'MRS',
 'MSAC',
 'MSLP',
 'MSSS',
 'MTCH',
 'MTL',
 'MTPP',
 'MTV',
 'MTY',
 'MUBA',
 'MUBY',
 'MUCA',
 'MUCL',
 'MUCM',
 'MUCU',
 'MUGM',
 'MUGT',
 'MUHA',
 'MUMO',
 'MUMZ',
 'MUNG',
 'MUVR',
 'MUVT',
 'MWCR',
 'MYBS',
 'MYEG',
 'MYGF',
 'MYGW',
 'MYL',
 'MYNN',
 'MZBZ',
 'MZT',
 'NCK',
 'NGX',
 'NHK',
 'NID',
 'NKX',
 'NOA',
 'NRU',
 'NTD',
 ...]
```

```
request.setLocationNames("KFRM")
cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)

try:
    fcstRun = DataAccessLayer.getForecastRun(cycles[-1], times)
    list(fcstRun)
    response = DataAccessLayer.getGeometryData(request,[fcstRun[0]])
except:
    print('No times available')
    exit
```

Model Sounding Parameters

Construct arrays for each parameter to plot (temperature, pressure, moisutre (spec. humidity), wind components, and cloud cover)

```
tmp,prs,sh = np.array([]),np.array([]),np.array([])
uc,vc,om,cld = np.array([]),np.array([]),np.array([]),np.array([])

for ob in response:
    tmp = np.append(tmp,ob.getNumber("temperature"))
    prs = np.append(prs,ob.getNumber("pressure"))
    sh = np.append(sh,ob.getNumber("specHum"))
    uc = np.append(uc,ob.getNumber("uComp"))
    vc = np.append(vc,ob.getNumber("vComp"))
    om = np.append(om,ob.getNumber("omega"))
    cld = np.append(cld,ob.getNumber("cldCvr"))

print("parms    = " + str(ob.getParameters()))
print("site     = " + str(ob.getLocationName()))
print("geom     = " + str(ob.getGeometry()))
print("datetime = " + str(ob.getDataTime()))
print("reftime  = " + str(ob.getDataTime().getRefTime()))
print("fcstHour = " + str(ob.getDataTime().getFcstTime()))
print("period   = " + str(ob.getDataTime().getValidPeriod()))
```

```
parms    = ['temperature', 'pressure', 'vComp', 'uComp', 'cldCvr', 'specHum', 'omega']
site     = KFRM
geom     = POINT (-94.41999816894531 43.65000152587891)
datetime = 2018-10-11 12:00:00
reftime  = Oct 11 18 12:00:00 GMT
fcstHour = 0
period   = (Oct 11 18 12:00:00 , Oct 11 18 12:00:00 )
```

Calculating Dewpoint from Specific Humidity

Because the modelsounding plugin does not return dewpoint values, we must calculate the profile ourselves. Here are three examples of dewpoint calculated from specific humidity, including a manual calculation following NCEP AWIPS/NSHARP.

1) MetPy calculated mixing ratio and vapor pressure

```
t = (tmp-273.15) * units.degC
p = prs/100 * units.mbar

u,v = uc*1.94384,vc*1.94384 # m/s to knots
spd = wind_speed(u, v) * units.knots
dir = wind_direction(u, v) * units.deg
```

```
rmix = (sh/(1-sh)) *1000 * units('g/kg')
e = vapor_pressure(p, rmix)
td = dewpoint(e)
```

2) metpy calculated assuming spec. humidity = mixing ratio

```
td2 = dewpoint(vapor_pressure(p, sh))
```

3) NCEP AWIPS soundingrequest plugin

based on GEMPAK/NSHARP, from https://github.com/Unidata/awips2-ncep/blob/unidata_16.2.2/edex/gov.noaa.nws.ncep.edex.plugin.soundingrequest/src/gov/noaa/nws/ncep/edex/plugin/soundingrequest/handler/MergeSounding.java#L1783

```
# new arrays
ntmp = tmp

# where p=pressure(pa), T=temp(C), T0=reference temp(273.16)
rh = 0.263*prs*sh / (np.exp(17.67*ntmp/(ntmp+273.15-29.65)))
vaps =  6.112 * np.exp((17.67 * ntmp) / (ntmp + 243.5))
vapr = rh * vaps / 100
dwpc = np.array(243.5 * (np.log(6.112) - np.log(vapr)) / (np.log(vapr) - np.log(6.
↪112) - 17.67)) * units.degC
```

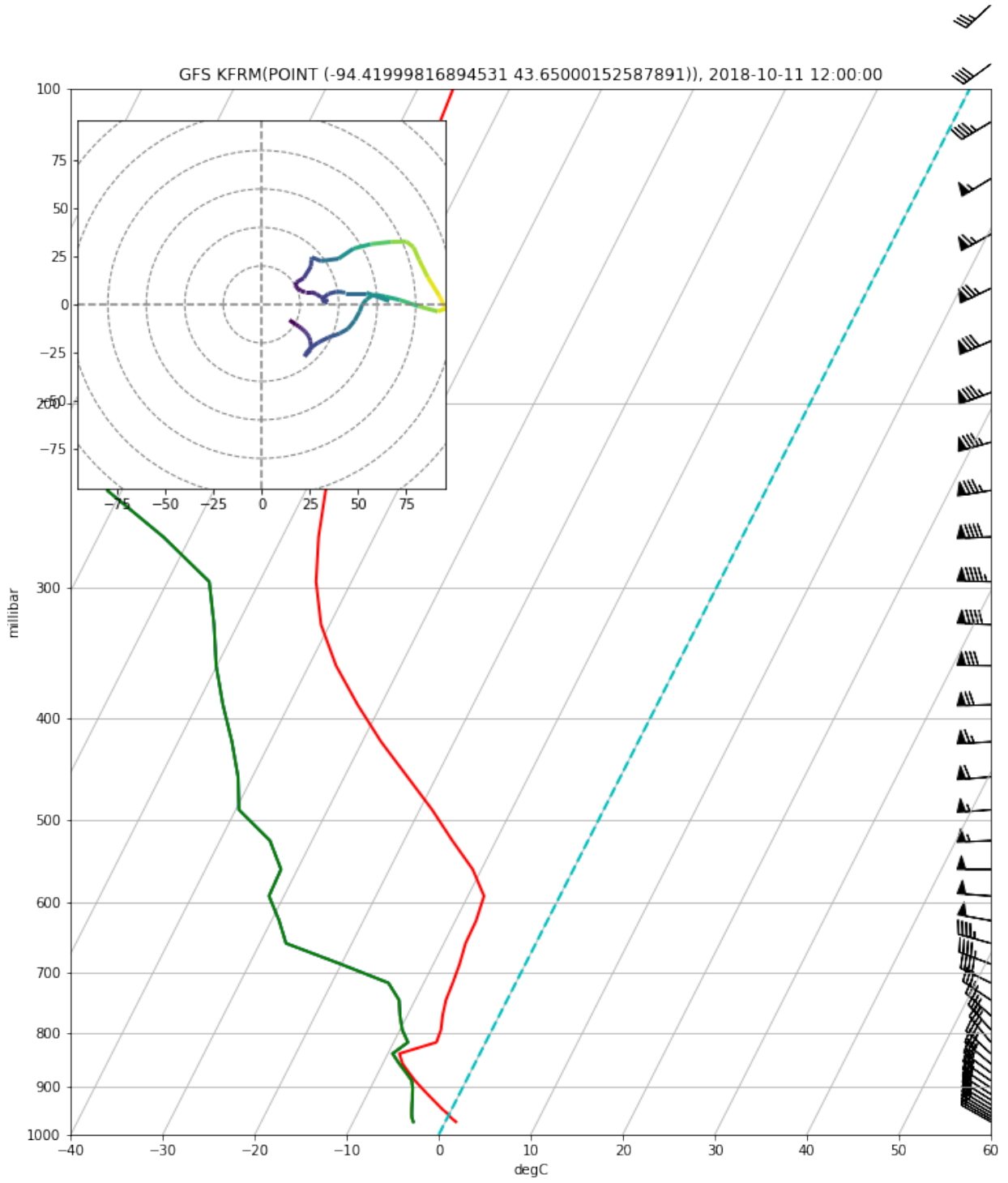
MetPy SkewT and Hodograph

```
%matplotlib inline

plt.rcParams['figure.figsize'] = (12, 14)

# Create a skewT plot
skew = SkewT()

# Plot the data
skew.plot(p, t, 'r', linewidth=2)
skew.plot(p, td, 'b', linewidth=2)
skew.plot(p, td2, 'y')
skew.plot(p, dwpc, 'g', linewidth=2)

skew.plot_barbs(p, u, v)
skew.ax.set_ylim(1000, 100)
skew.ax.set_xlim(-40, 60)

plt.title( forecastModel + " " \
          + ob.getLocationName() \
          + "("+ str(ob.getGeometry()) + ")" \
          + ", " + str(ob.getDataTime())
)
```

(continues on next page)

(continued from previous page)

```
# An example of a slanted line at constant T -- in this case the 0 isotherm
l = skew.ax.axvline(0, color='c', linestyle='--', linewidth=2)

# Draw hodograph
ax_hod = inset_axes(skew.ax, '40%', '40%', loc=2)
h = Hodograph(ax_hod, component_range=wind_speed(u, v).max())
h.add_grid(increment=20)
h.plot_colormapped(u, v, spd)

# Show the plot
plt.show()
```



NEXRAD Level3 Radar

Notebook

```
import warnings
```

(continues on next page)

(continued from previous page)

```
from awips.dataaccess import DataAccessLayer
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import numpy as np
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER

DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest("radar")
available_locs = DataAccessLayer.getAvailableLocationNames(request)
available_locs.sort()
list(available_locs)
request.setLocationNames("kmhx")
availableParms = DataAccessLayer.getAvailableParameters(request)
availableParms.sort()
#list(availableParms)

productIDs = DataAccessLayer.getRadarProductIDs(availableParms)
productNames = DataAccessLayer.getRadarProductNames(availableParms)
print(productIDs)
print(productNames)
```

```
['134', '135', '138', '141', '159', '161', '163', '165', '166', '169', '170', '171',
↪'172', '173', '174', '175', '176', '177', '19', '20', '27', '32', '37', '41', '56',
↪'57', '58', '78', '80', '81', '94', '99']
['Composite Refl', 'Correlation Coeff', 'Diff Reflectivity', 'Digital Hybrid Scan Refl
↪', 'Digital Inst Precip Rate', 'Digital Precip Array', 'Digital Vert Integ Liq',
↪'Echo Tops', 'Enhanced Echo Tops', 'Hybrid Hydrometeor Class', 'Hydrometeor Class',
↪'Melting Layer', 'Mesocyclone', 'One Hour Accum', 'One Hour Diff', 'One Hour Precip
↪', 'One Hour Unbiased Accum', 'Reflectivity', 'Specific Diff Phase', 'Storm Rel␣
↪Velocity', 'Storm Total Accum', 'Storm Total Diff', 'Storm Total Precip', 'Storm␣
↪Track', 'User Select Accum', 'Velocity', 'Vert Integ Liq']
```

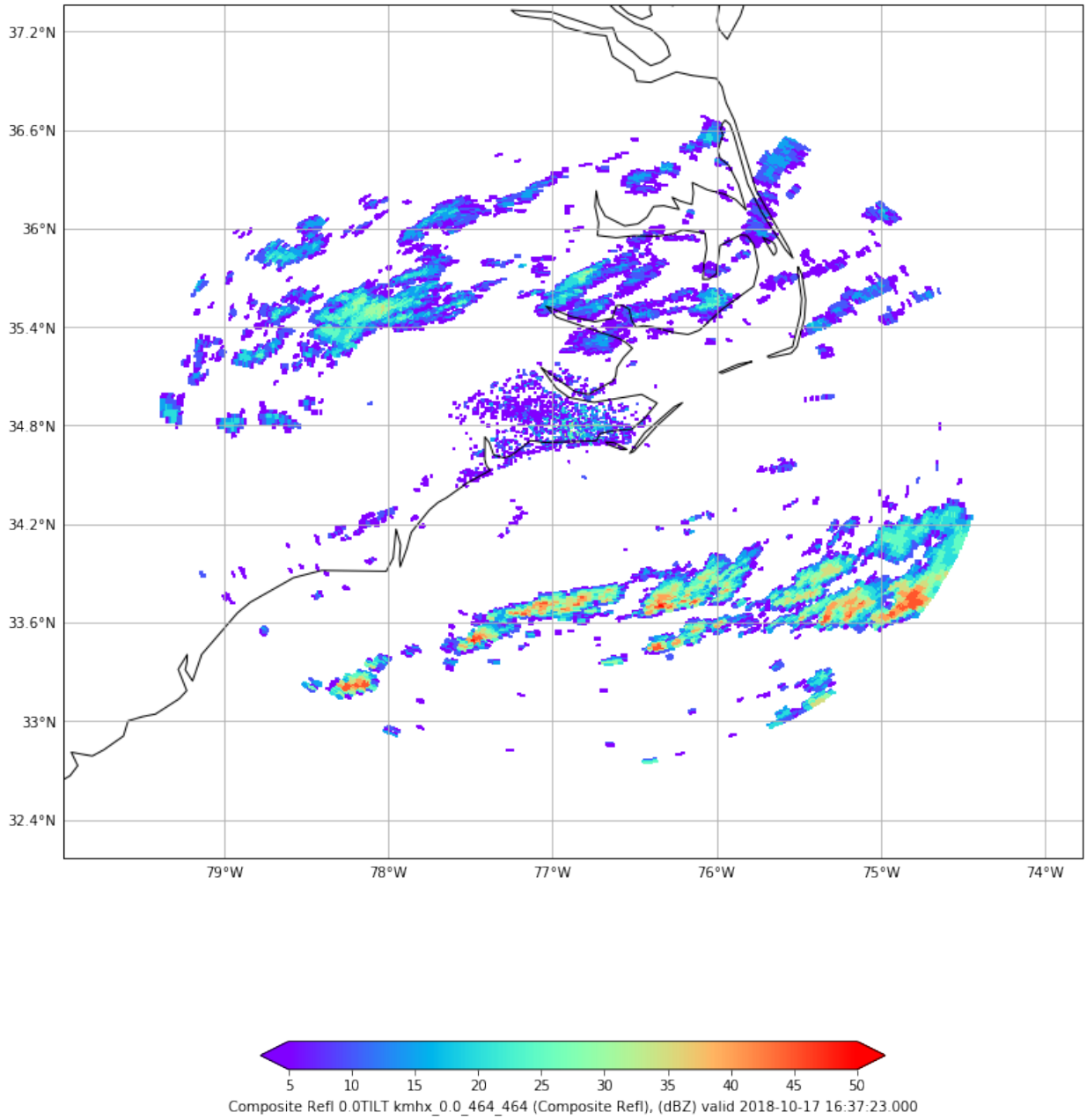
```
warnings.filterwarnings("ignore",category =RuntimeWarning)

def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(16, 16),
                           subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax

nexrad_data = {}

for prod in productNames:

    request.setParameters(prod)
    availableLevels = DataAccessLayer.getAvailableLevels(request)
    if availableLevels:
        request.setLevels(availableLevels[0])
    else:
        print("No levels found for " + prod)
        continue
```

(continues on next page)

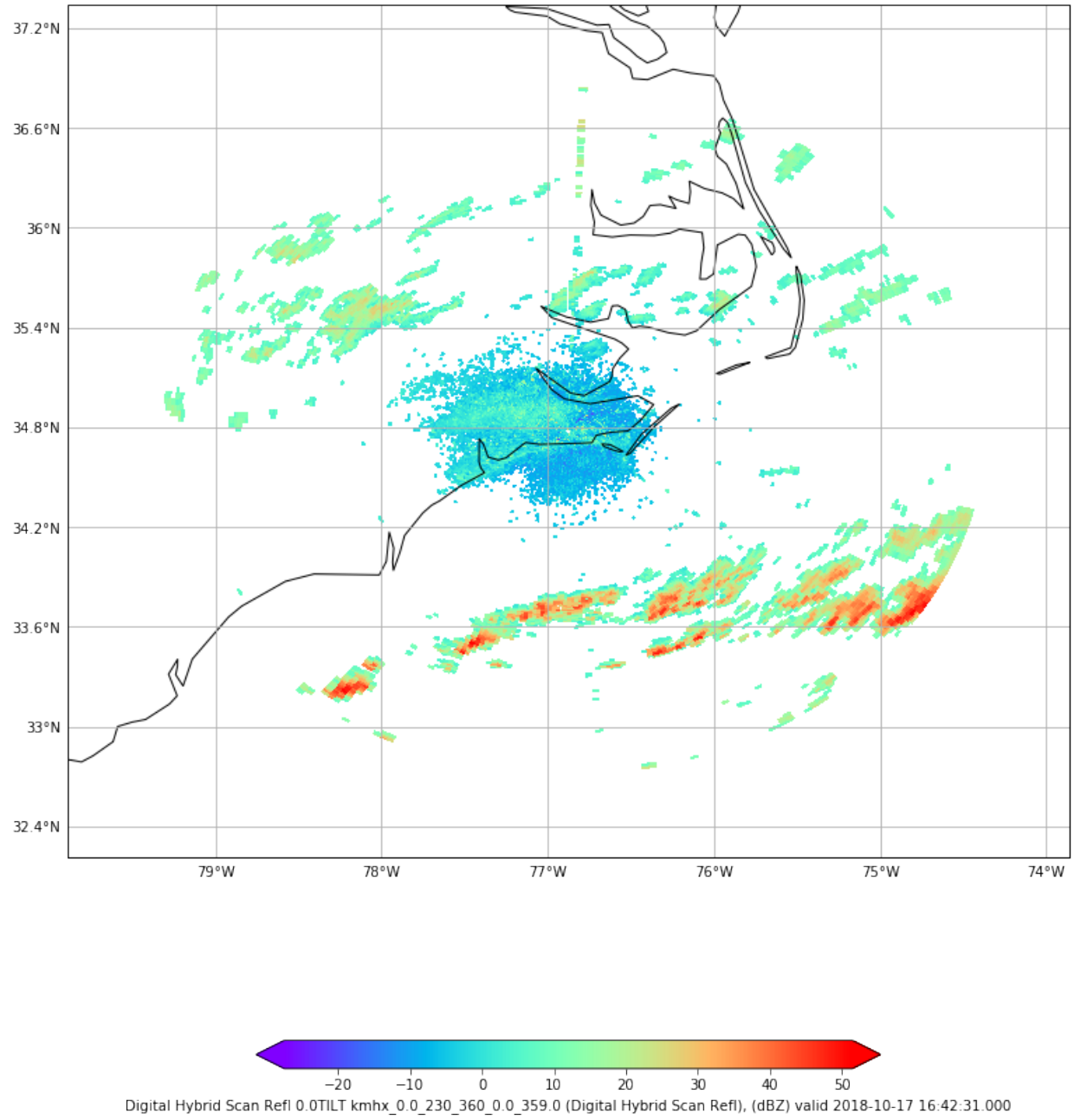
(continued from previous page)

```
    cycles = DataAccessLayer.getAvailableTimes(request, True)
    times = DataAccessLayer.getAvailableTimes(request)

    if times:
        print()
        response = DataAccessLayer.getGridData(request, [times[-1]])
        print("Recs : ", len(response))

        if response:
            grid = response[0]
        else:
            continue
        data = grid.getRawData()
        lons, lats = grid.getLatLonCoords()

        nexrad_data[prod] = data

        print('Time :', str(grid.getDataTime()))
        flat = np.ndarray.flatten(data)
        print('Name :', str(grid.getLocationName()))
        print('Prod :', str(grid.getParameter()))
        print('Range:' , np.nanmin(flat), " to ", np.nanmax(flat), " (Unit :", grid.
↪getUnit(), ")")
        print('Size :', str(data.shape))
        print()

        cmap = plt.get_cmap('rainbow')
        bbox = [lons.min()-0.5, lons.max()+0.5, lats.min()-0.5, lats.max()+0.5]
        fig, ax = make_map(bbox=bbox)
        cs = ax.pcolormesh(lons, lats, data, cmap=cmap)
        cbar = fig.colorbar(cs, extend='both', shrink=0.5, orientation='horizontal')
        cbar.set_label(grid.getParameter() +" " + grid.getLevel() + " " \
                       + grid.getLocationName() + " (" + prod + "), (" + grid.
↪getUnit() + ") " \
                       + "valid " + str(grid.getDataTime().getRefTime()))
        plt.show()
```

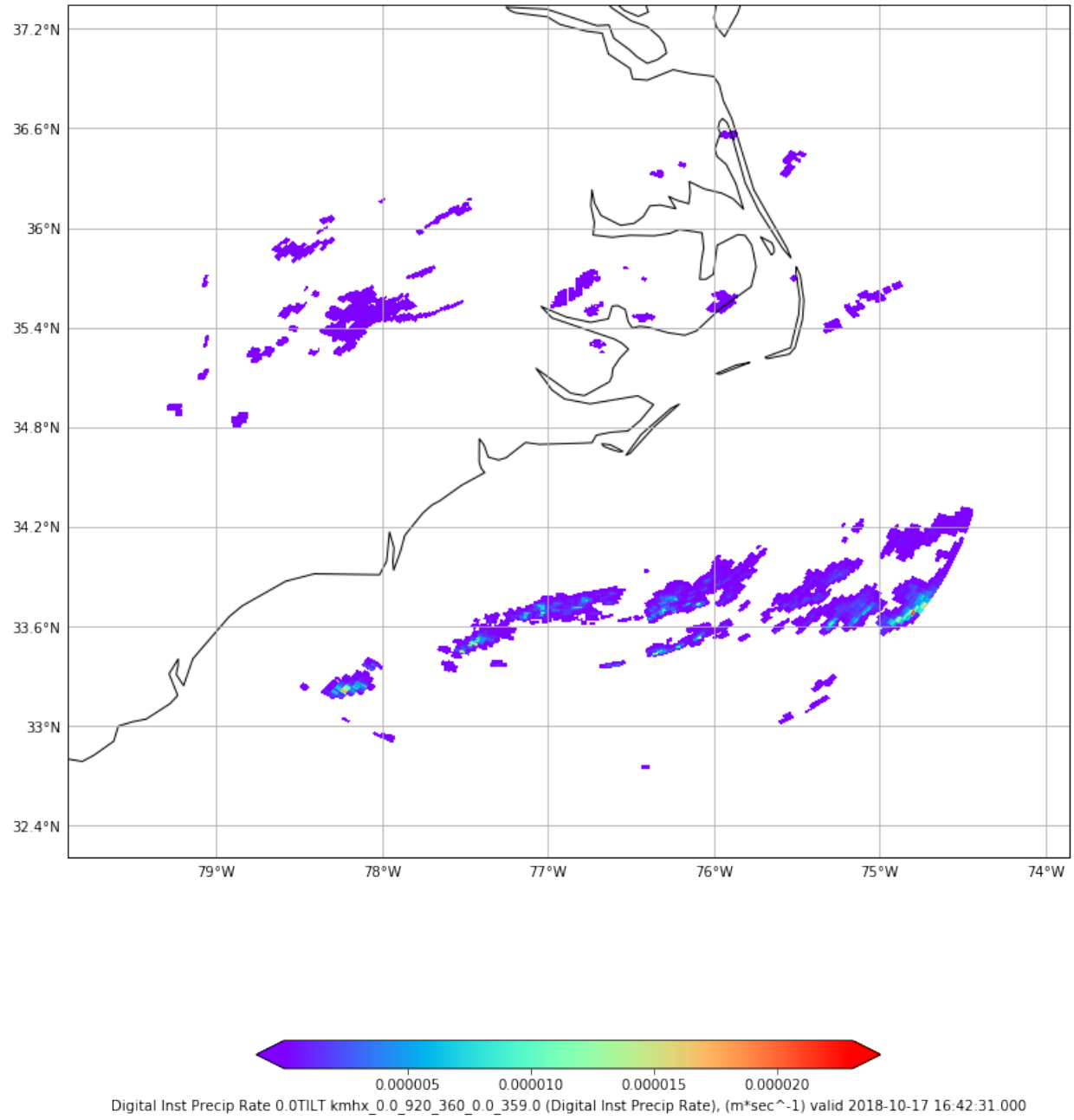
```
Recs :  1
Time : 2018-10-17 16:37:23
Name : kmhx_0.0_464_464
Prod : Composite Refl
Range: 5.0  to  50.0  (Unit : dBZ )
Size : (464, 464)
```



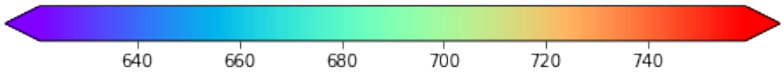

```
No levels found for Correlation Coeff
No levels found for Diff Reflectivity

Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_230_360_0.0_359.0
Prod : Digital Hybrid Scan Refl
Range: -27.5  to  51.5  (Unit : dBZ )
Size : (230, 360)
```




```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : Digital Inst Precip Rate
Range: 7.0555557e-09  to  2.3071667e-05  (Unit : m*sec^-1 )
Size : (920, 360)
```

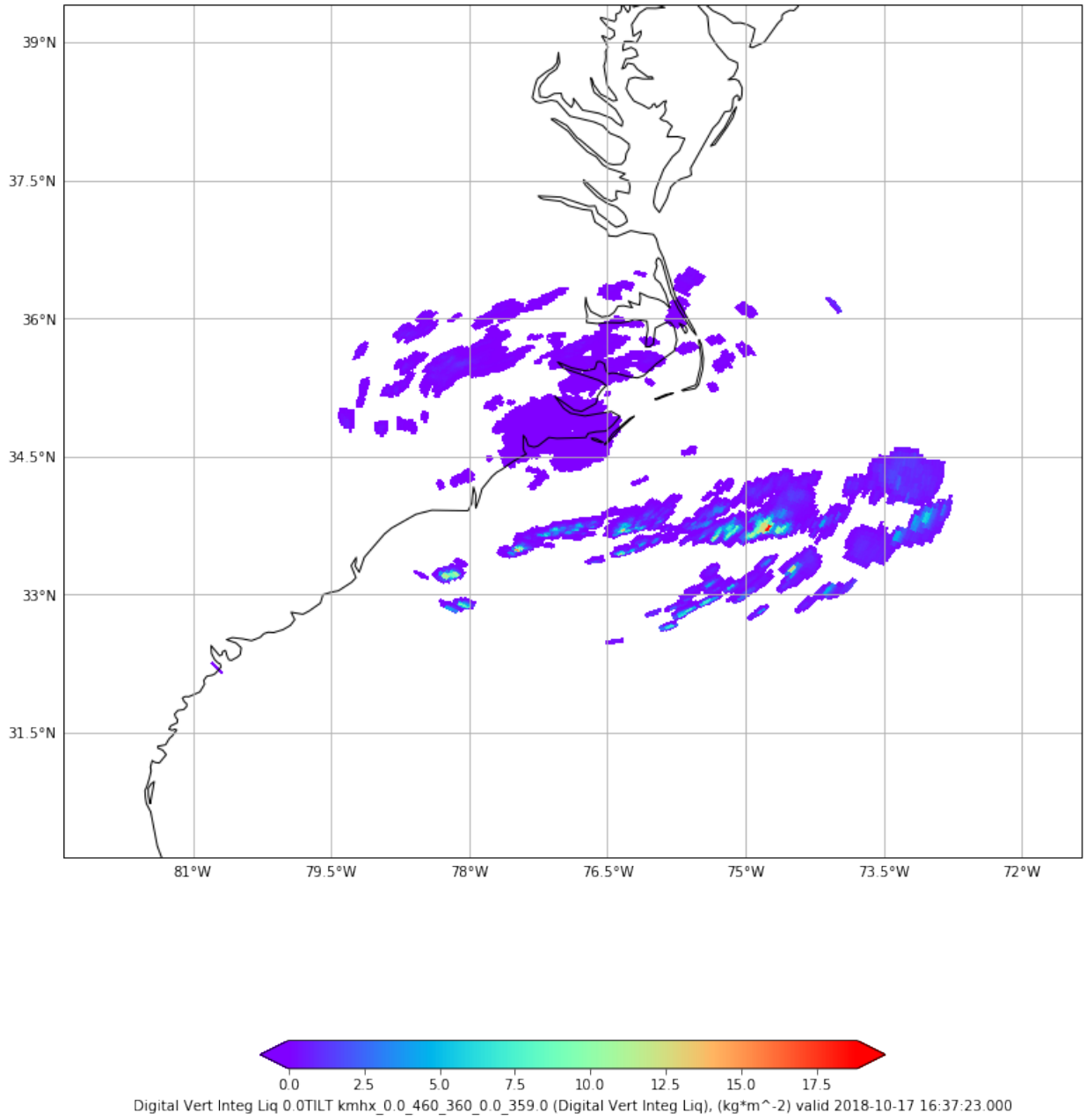



```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_13_13
Prod : Digital Precip Array
Range: 190.0  to  690.0  (Unit : count )
Size : (13, 13)
```

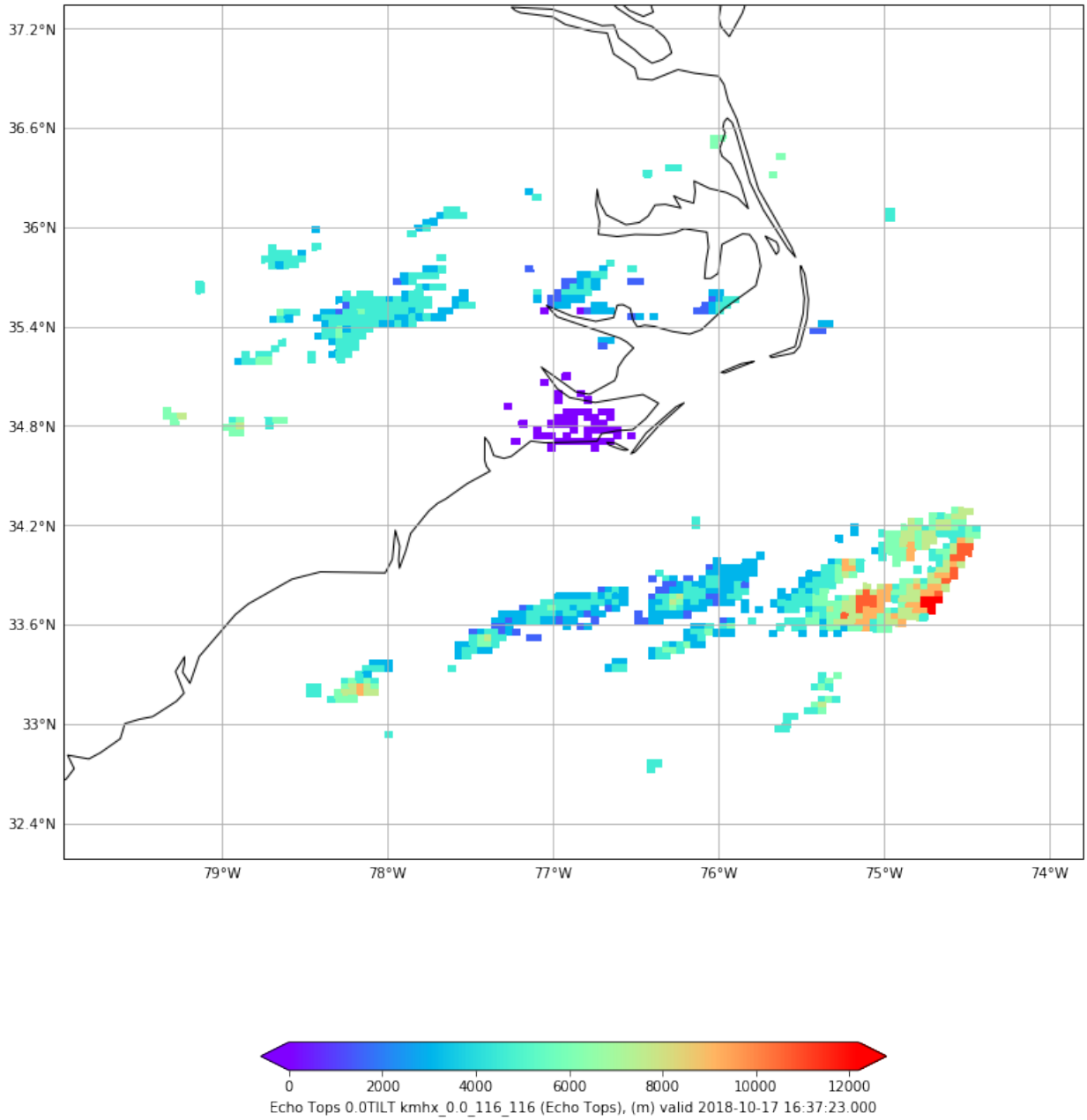


Digital Precip Array 0.0TILT kmhx_0.0_13_13 (Digital Precip Array), (count) valid 2018-10-17 16:42:31.000

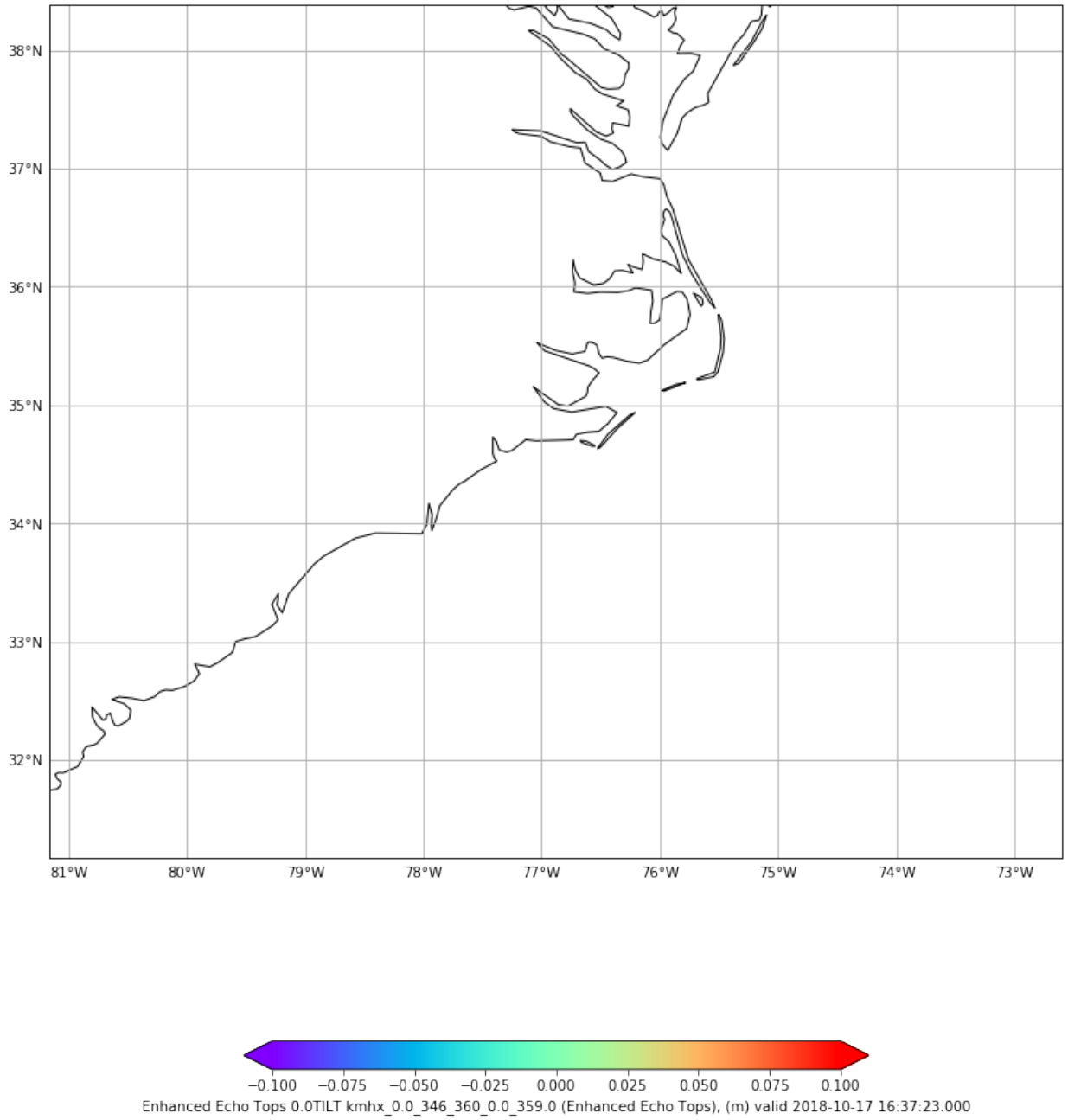
```
Recs :  1
Time : 2018-10-17 16:37:23
Name : kmhx_0.0_460_360_0.0_359.0
Prod : Digital Vert Integ Liq
Range: 0.0  to  18.834518  (Unit : kg*m^-2 )
Size : (460, 360)
```



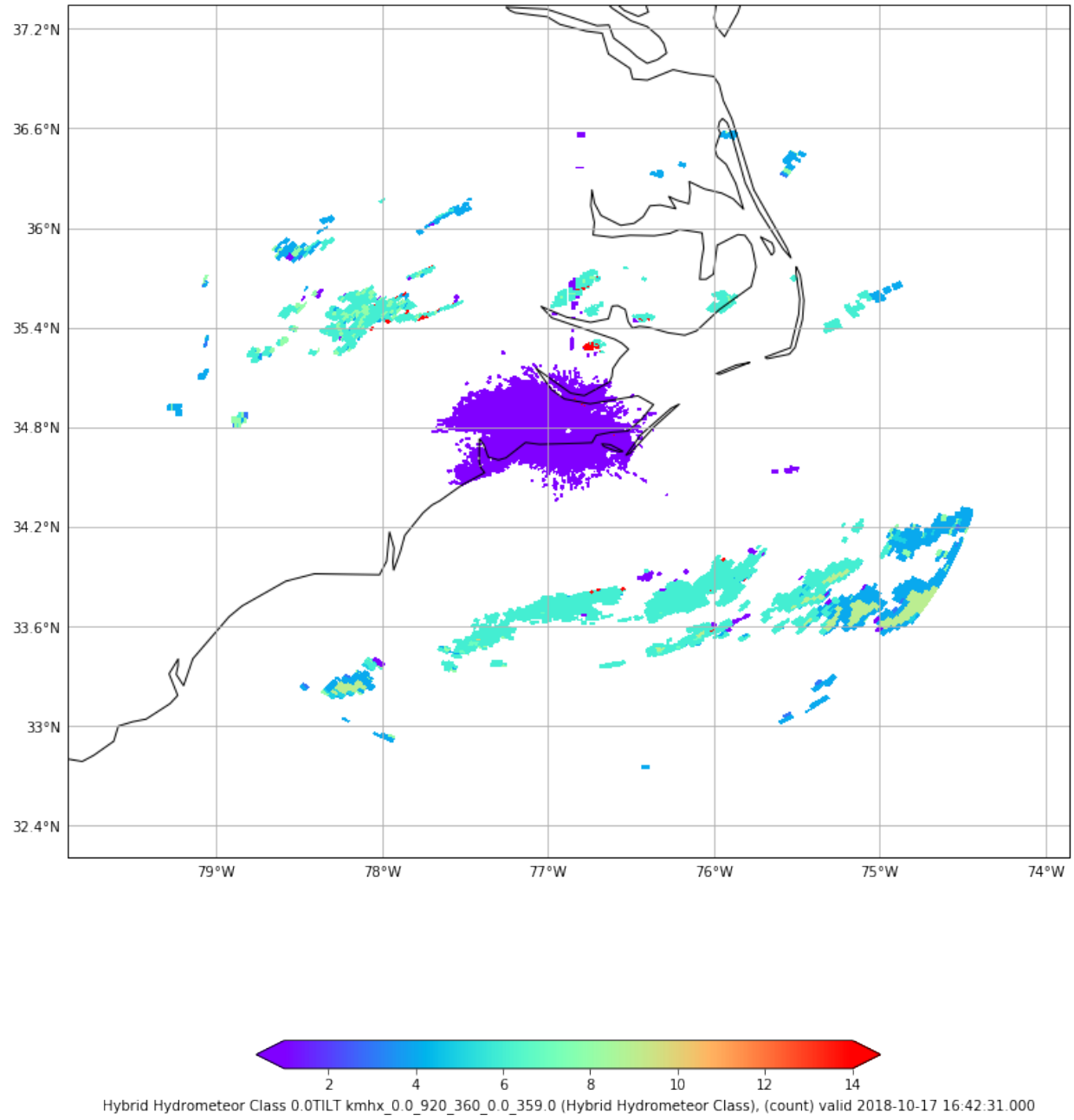

```
Recs :  1
Time : 2018-10-17 16:37:23
Name : kmhx_0.0_116_116
Prod : Echo Tops
Range: 0.0  to  12192.0  (Unit : m )
Size : (116, 116)
```



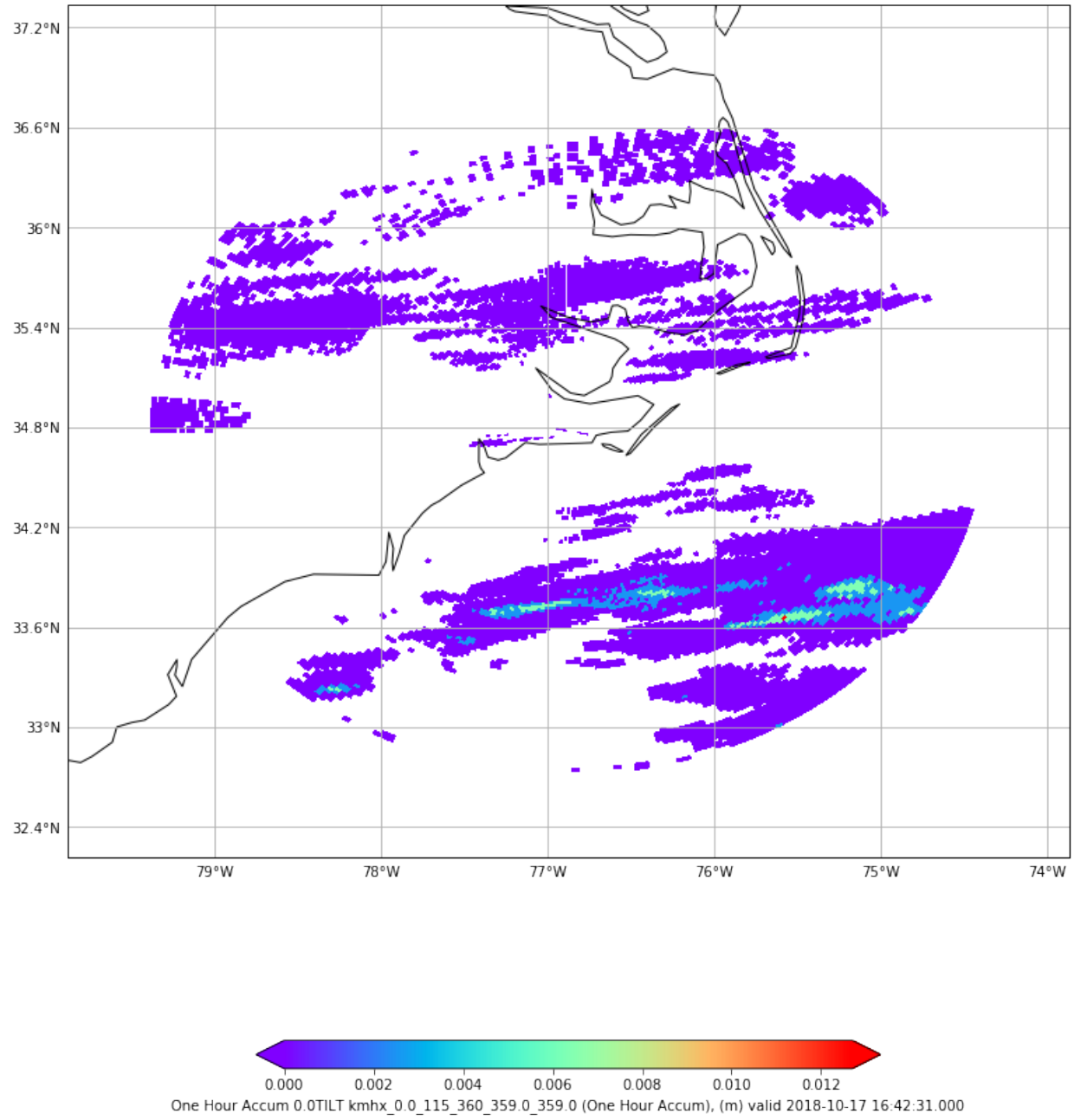

```
Recs :  1
Time : 2018-10-17 16:37:23
Name : kmhx_0.0_346_360_0.0_359.0
Prod : Enhanced Echo Tops
Range: nan  to  nan  (Unit : m )
Size : (346, 360)
```



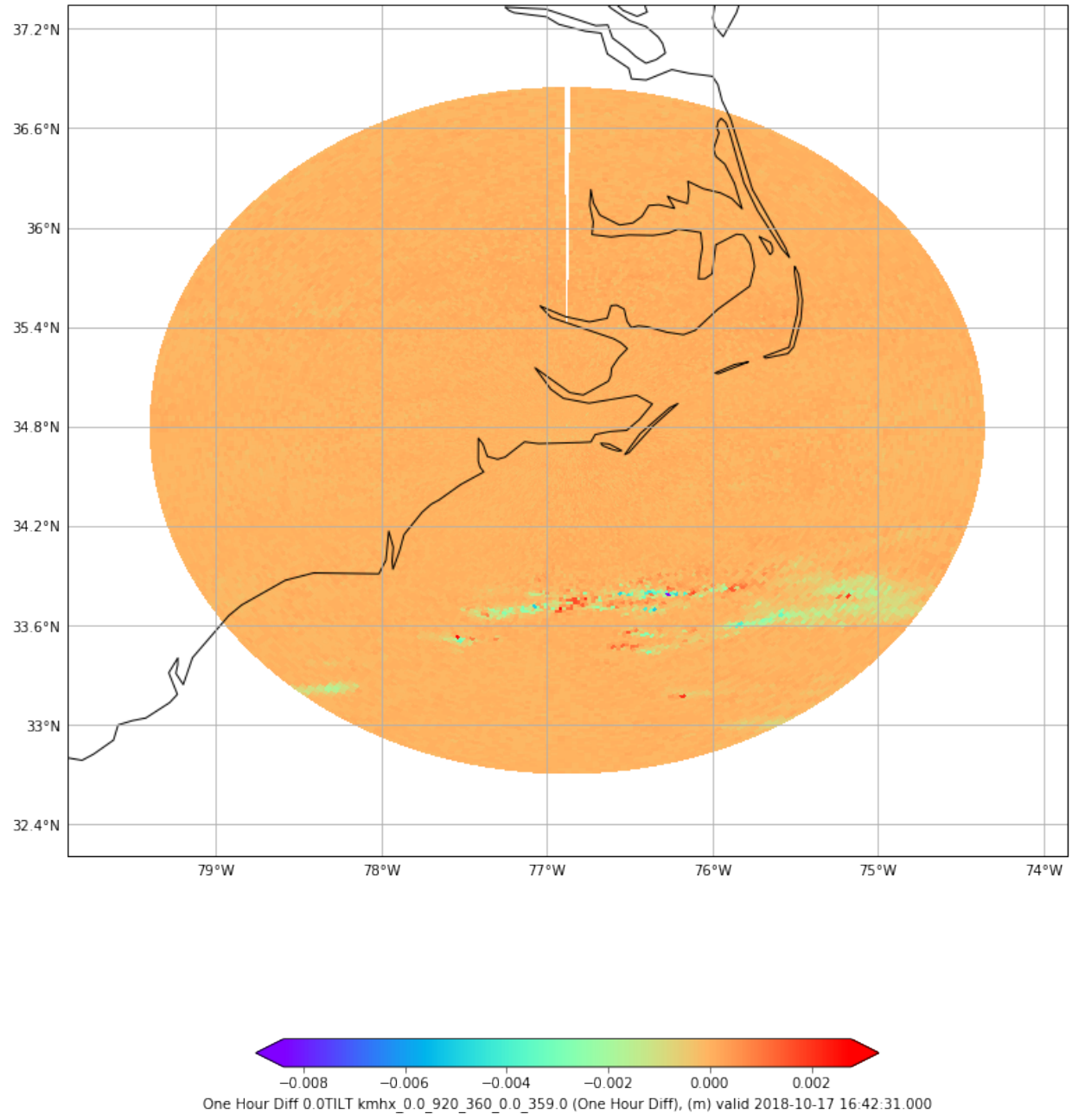

```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : Hybrid Hydrometeor Class
Range: 1.0  to  14.0  (Unit : count )
Size : (920, 360)
```



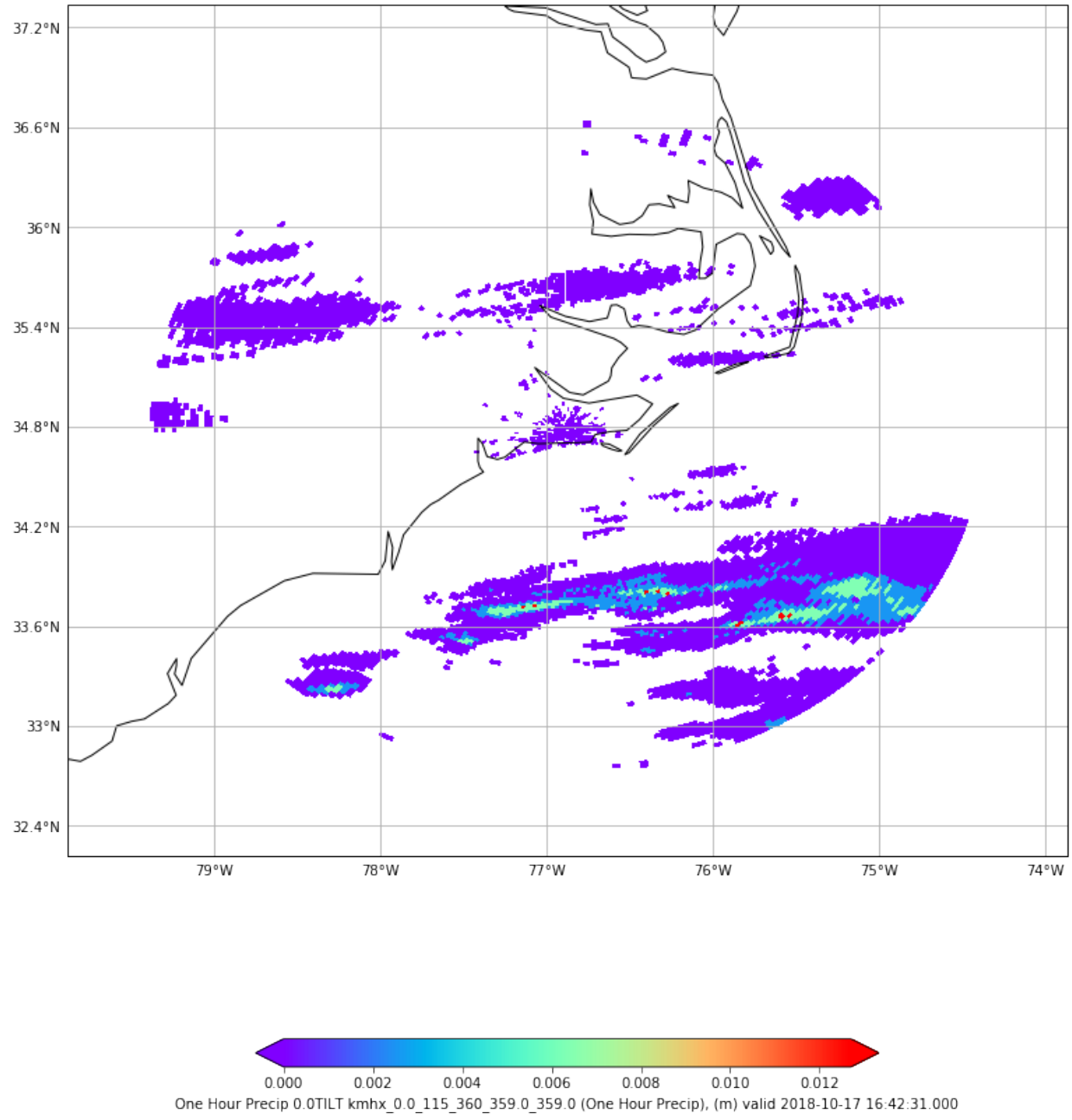

```
No levels found for Hydrometeor Class
No levels found for Melting Layer

Recs :  0

Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_115_360_359.0_359.0
Prod : One Hour Accum
Range: 0.0  to  0.0127  (Unit : m )
Size : (115, 360)
```



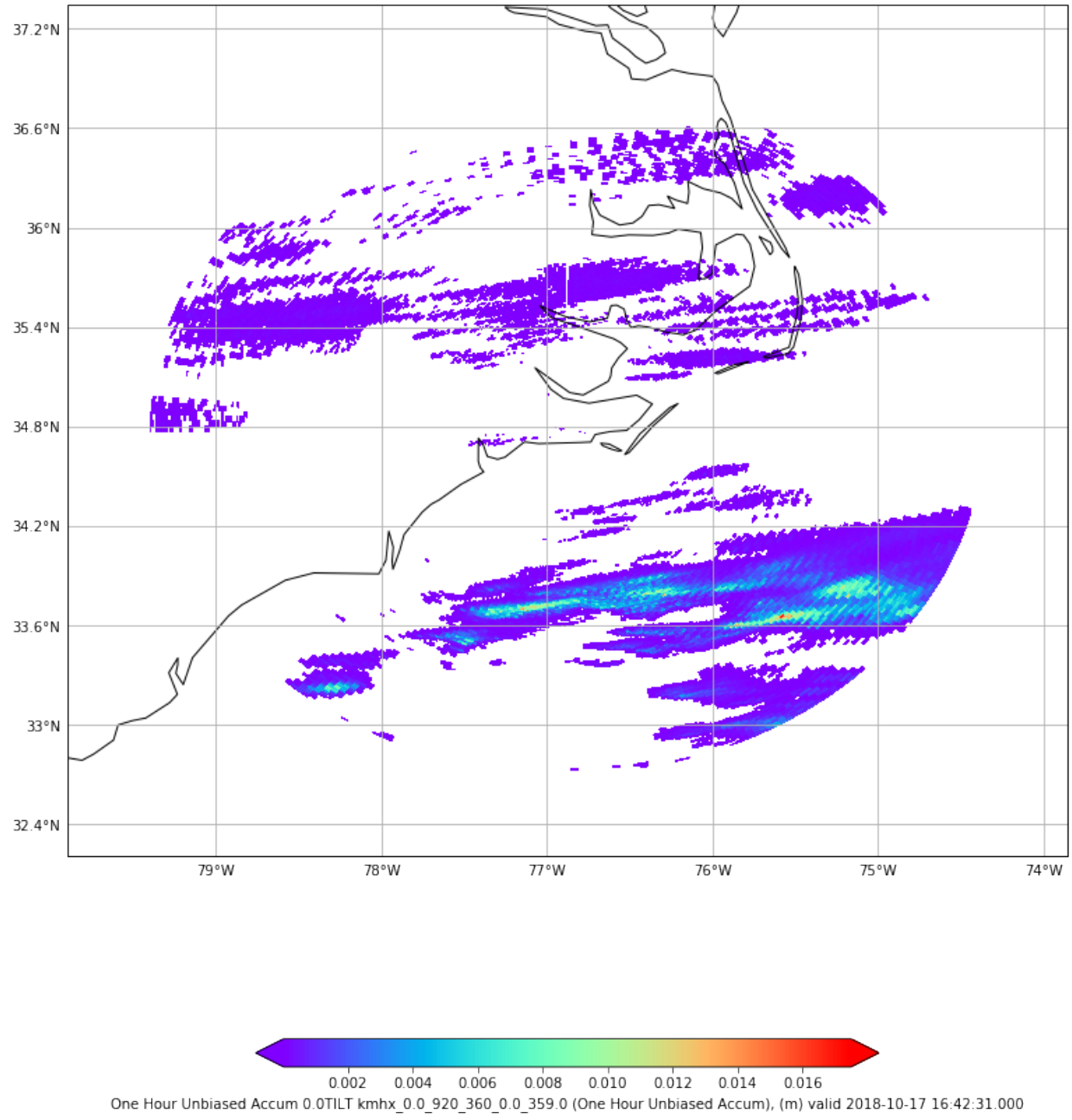

```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : One Hour Diff
Range: -0.008382  to  0.0027720002  (Unit : m )
Size : (920, 360)
```



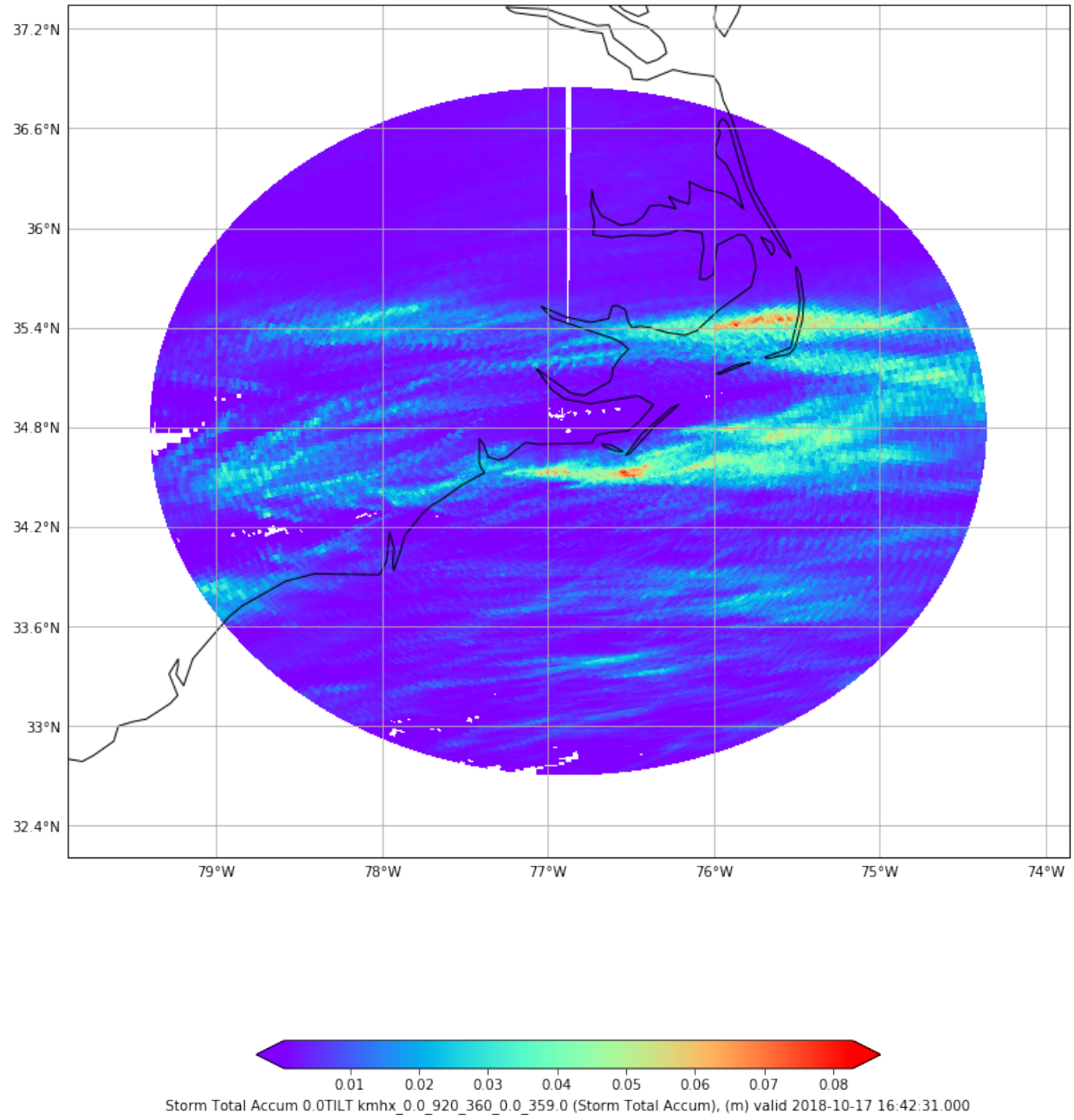

```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_115_360_359.0_359.0
Prod : One Hour Precip
Range: 0.0  to  0.0127  (Unit : m )
Size : (115, 360)
```



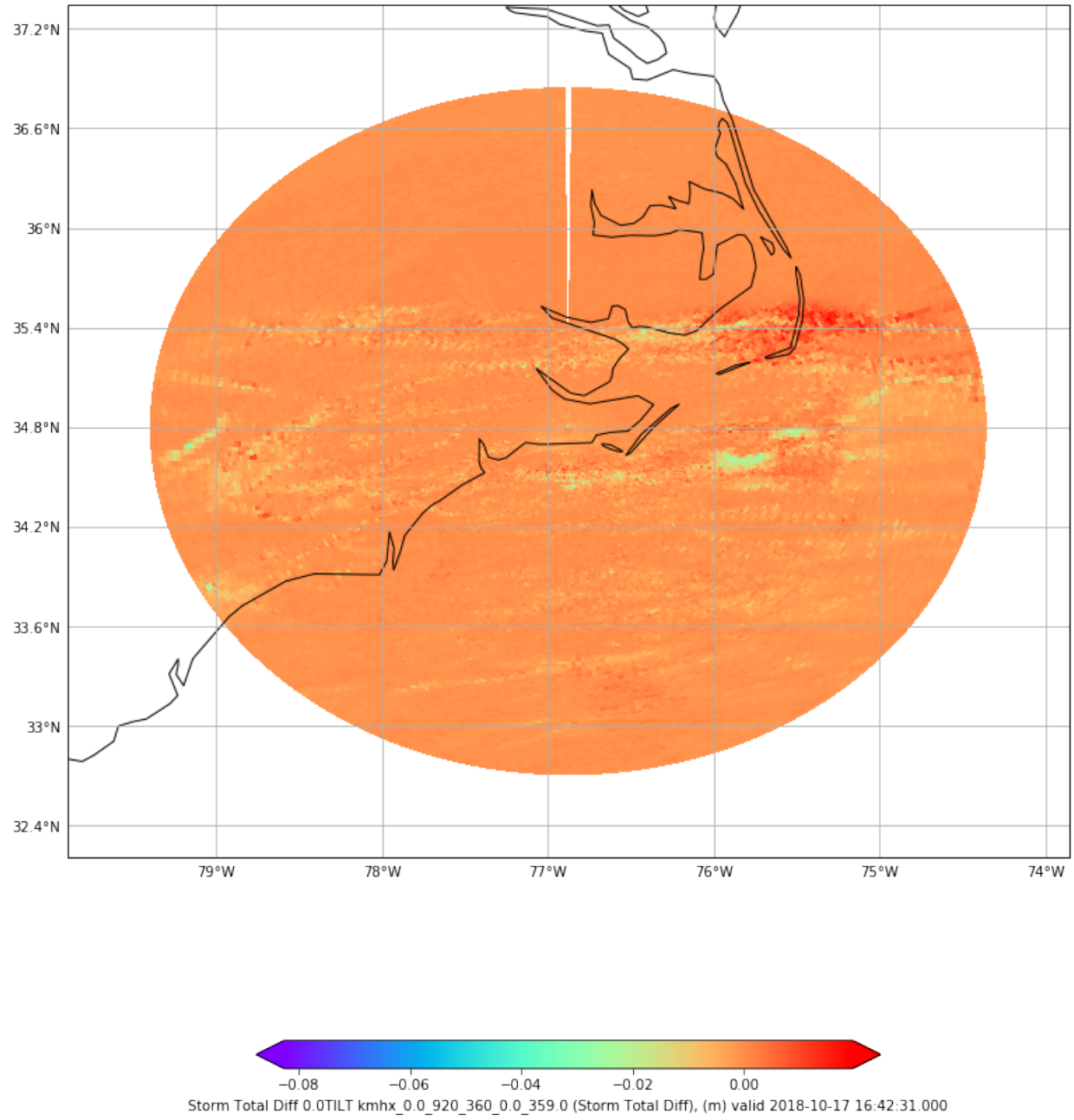

```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : One Hour Unbiased Accum
Range: 2.5775646e-05  to  0.017472787  (Unit : m )
Size : (920, 360)
```



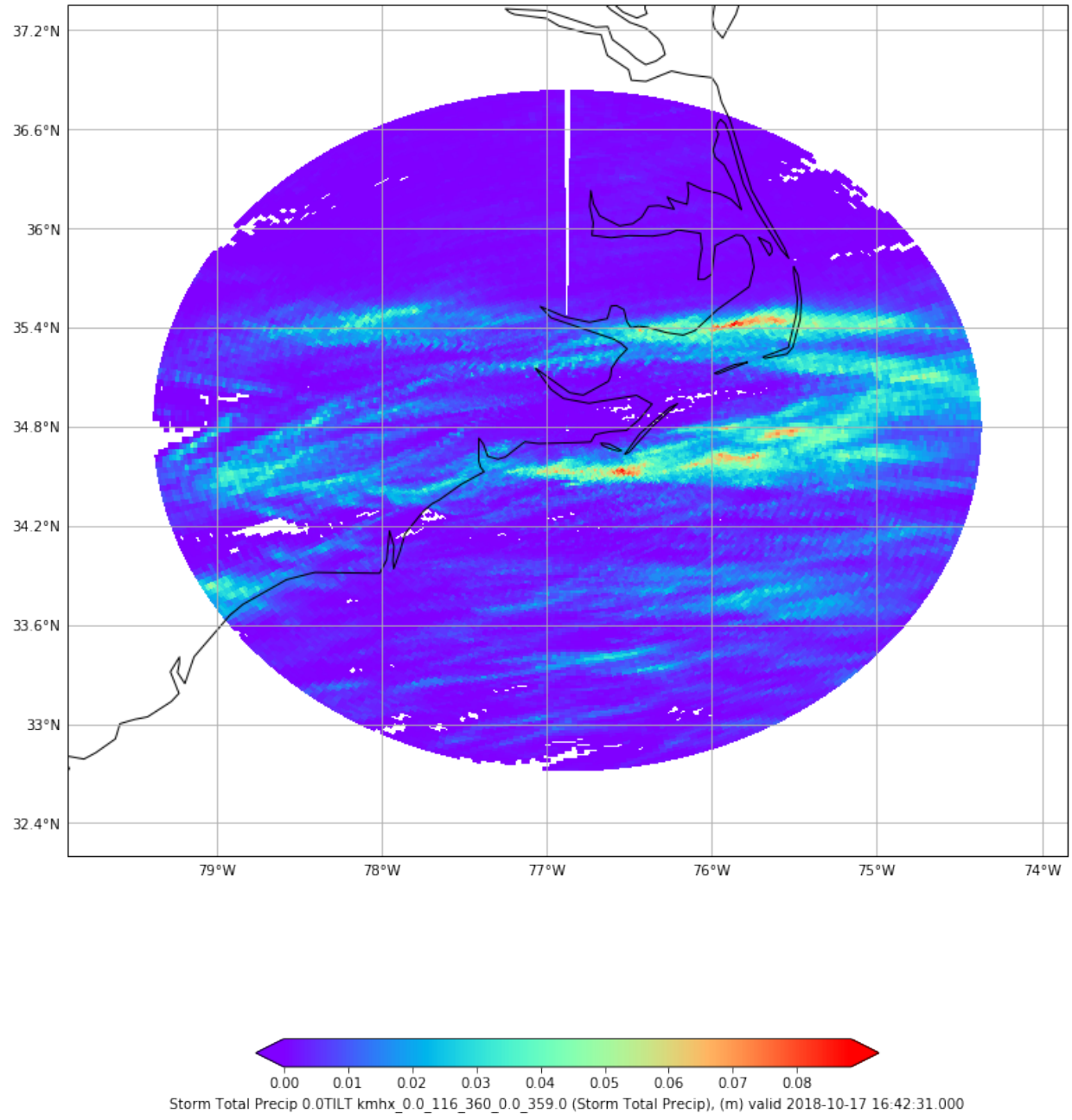

```
No levels found for Reflectivity
No levels found for Specific Diff Phase
No levels found for Storm Rel Velocity

Recs :  2
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : Storm Total Accum
Range: 0.000508  to  0.082804  (Unit : m )
Size : (920, 360)
```



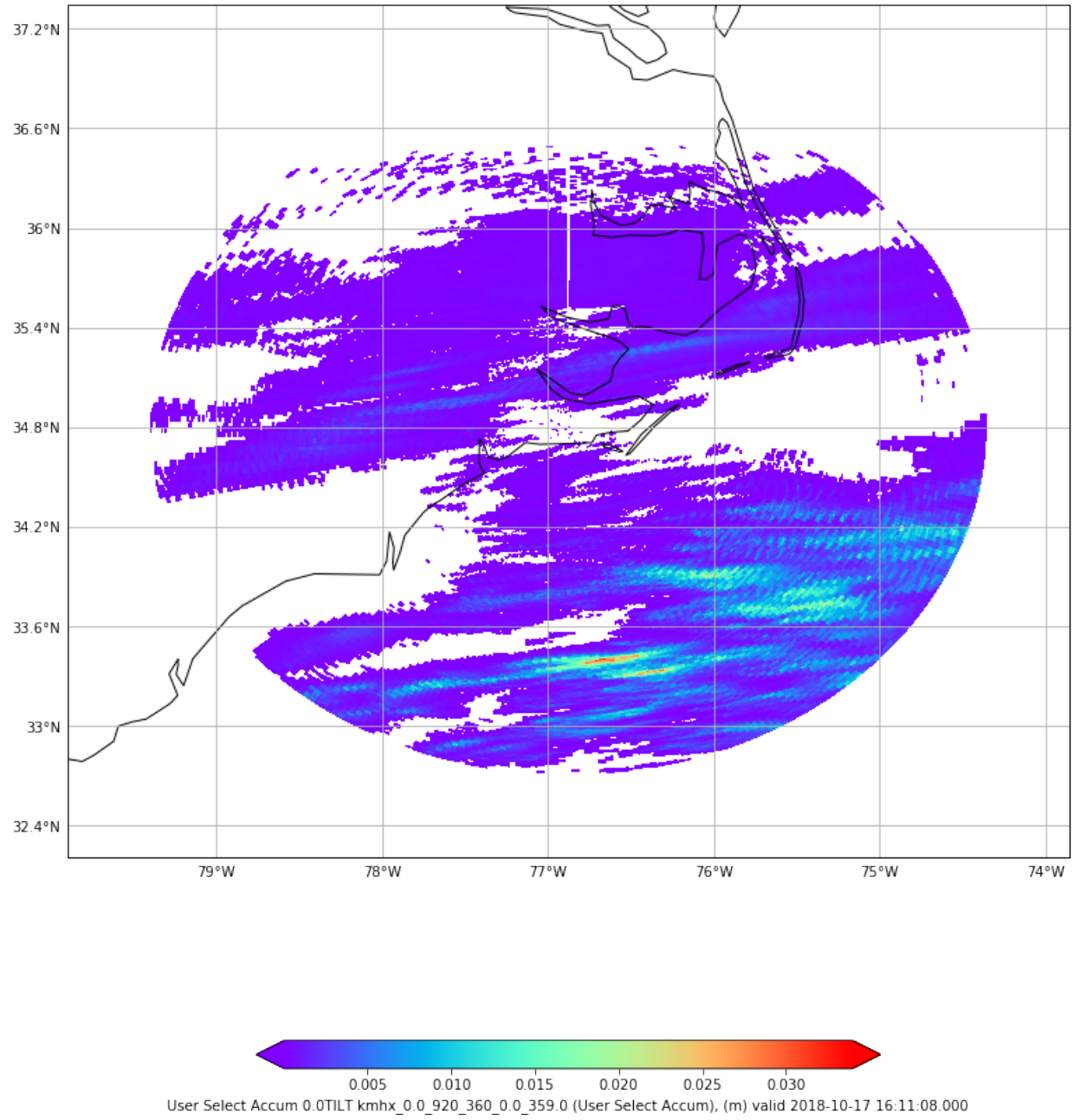

```
Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_920_360_0.0_359.0
Prod : Storm Total Diff
Range: -0.08255  to  0.019499999  (Unit : m )
Size : (920, 360)
```



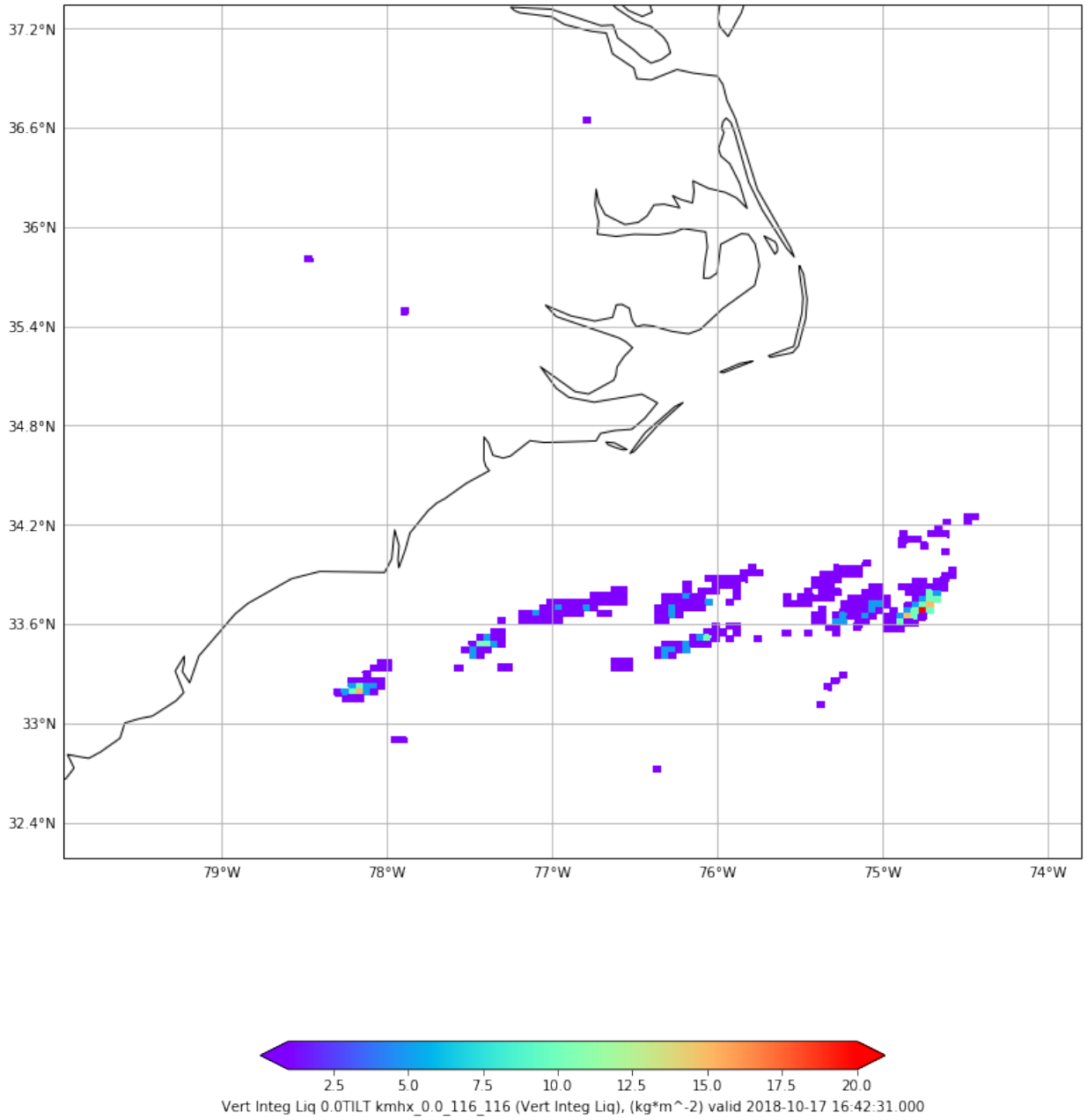

```
Recs :  2
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_116_360_0.0_359.0
Prod : Storm Total Precip
Range: 0.0  to  0.088392  (Unit : m )
Size : (116, 360)
```




```
Recs :  0

Recs :  1
Time : 2018-10-17 16:11:08
Name : kmhx_0.0_920_360_0.0_359.0
Prod : User Select Accum
Range: 2.5399999e-05  to  0.033959802  (Unit : m )
Size : (920, 360)
```




```
No levels found for Velocity

Recs :  1
Time : 2018-10-17 16:42:31
Name : kmhx_0.0_116_116
Prod : Vert Integ Liq
Range: 1.0  to  20.0  (Unit : kg*m^-2 )
Size : (116, 116)
```



Precip Accumulation-Region Of Interest

Notebook

A way to determine areas of greatest precipitation and generate imagery for that sector.

```
from awips.dataaccess import DataAccessLayer
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
from metpy.units import units
import numpy as np
```

(continues on next page)

(continued from previous page)

```
from shapely.geometry import Point, Polygon

%matplotlib inline

conus=[-120, -65, 28, 50]
conus_envelope = Polygon([(conus[0],conus[2]),(conus[0],conus[3]),
                          (conus[1],conus[3]),(conus[1],conus[2]),
                          (conus[0],conus[2])])

DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest("grid", envelope=conus_envelope)
request.setLocationNames("NAM40")
request.setLevels("0.0SFC")
request.setParameters("TP")

cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)
fcstRun = DataAccessLayer.getForecastRun(cycles[-2], times)
```

Calculate accumulated precipitation

```
for i, tt in enumerate(fcstRun):
    response = DataAccessLayer.getGridData(request, [tt])
    grid = response[0]
    if i>0:
        data += grid.getRawData()
    else:
        data = grid.getRawData()
    data[data <= -9999] = 0
    print(data.min(), data.max(), grid.getDataTime().getFcstTime()/3600)


lons, lats = grid.getLatLonCoords()
bbox = [lons.min(), lons.max(), lats.min(), lats.max()]
fcstHr = int(grid.getDataTime().getFcstTime()/3600)

tp_inch = data * (0.0393701)
print(tp_inch.min(), tp_inch.max())
```

```
0.0 0.0 0.0
0.0 32.1875 3.0
0.0 52.125 6.0
0.0 74.375 9.0
0.0 77.125 12.0
0.0 78.625 15.0
0.0 78.75 18.0
0.0 78.75 21.0
0.0 79.375 24.0
0.0 82.25 27.0
0.0 84.0 30.0
0.0 84.6875 33.0
0.0 85.625 36.0
0.0 87.3125 39.0
0.0 87.75 42.0
0.0 87.75 45.0
0.0 89.375 48.0
```

(continues on next page)

(continued from previous page)

```
0.0 127.875 51.0
0.0 139.5625 54.0
0.0 139.6875 57.0
0.0 140.5625 60.0
0.0 140.625 63.0
0.0 140.625 66.0
0.0 140.625 69.0
0.0 140.625 72.0
0.0 140.625 75.0
0.0 140.625 78.0
0.0 140.625 81.0
0.0 140.625 84.0
0.0 5.5364203
```

Determine lat/lon of maximum rainfall value:

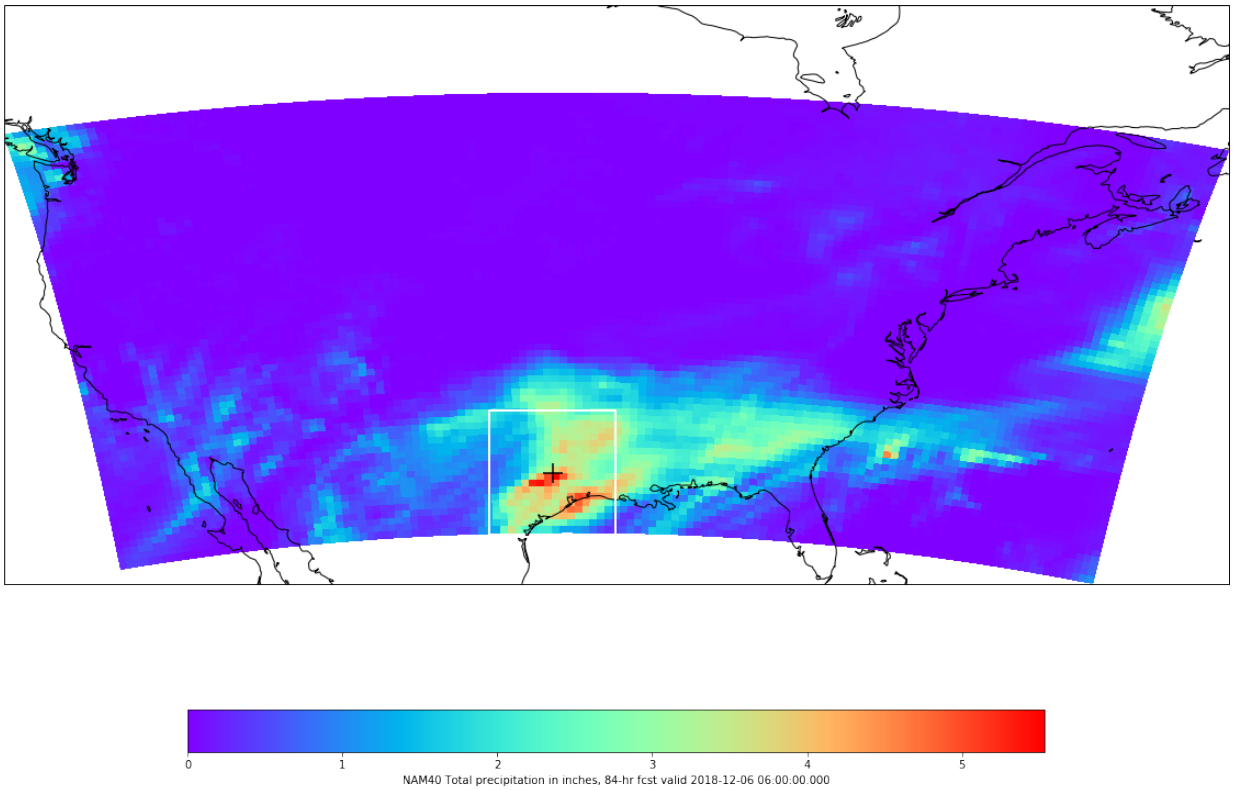
```
ii,jj = np.where(tp_inch==tp_inch.max())
i=ii[0]
j=jj[0]
point = Point(lons[i][j], lats[i][j])
```

Draw CONUS map

```
def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(20, 14),
             subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    return fig, ax

cmap = plt.get_cmap('rainbow')
fig, ax = make_map(bbox=bbox)
cs = ax.pcolormesh(lons, lats, tp_inch, cmap=cmap)
cbar = fig.colorbar(cs, shrink=0.7, orientation='horizontal')
cbar.set_label(grid.getLocationName() + " Total precipitation in inches, " \
               + str(fcstHr) + "-hr fcst valid " + str(grid.getDataTime().
↪getRefTime()))

ax.scatter(point.x, point.y, s=300,
           transform=ccrs.Geodetic(),marker="+",facecolor='black')

inc = 3.5
box=[point.x-inc, point.x+inc, point.y-inc, point.y+inc]
polygon = Polygon([(box[0],box[2]),(box[0],box[3]),
                   (box[1],box[3]),(box[1],box[2]),
                   (box[0],box[2])])
ax.add_geometries([polygon], ccrs.Geodetic(), facecolor='none', edgecolor='white',␣
↪linewidth=2)
```

```
<cartopy.mpl.feature_artist.FeatureArtist at 0x11b971da0>
```

```
/Users/mjames/miniconda3/envs/python3-awips/lib/python3.6/site-packages/cartopy/mpl/
↪geoaxes.py:623: UserWarning: Approximating coordinate system <cartopy._crs.Geodetic␣
↪object at 0x11b9e10f8> with the PlateCarree projection.
  'PlateCarree projection.'.format(crs))
```



Now create a new gridded data request with a geometry envelope for our Region of Interest

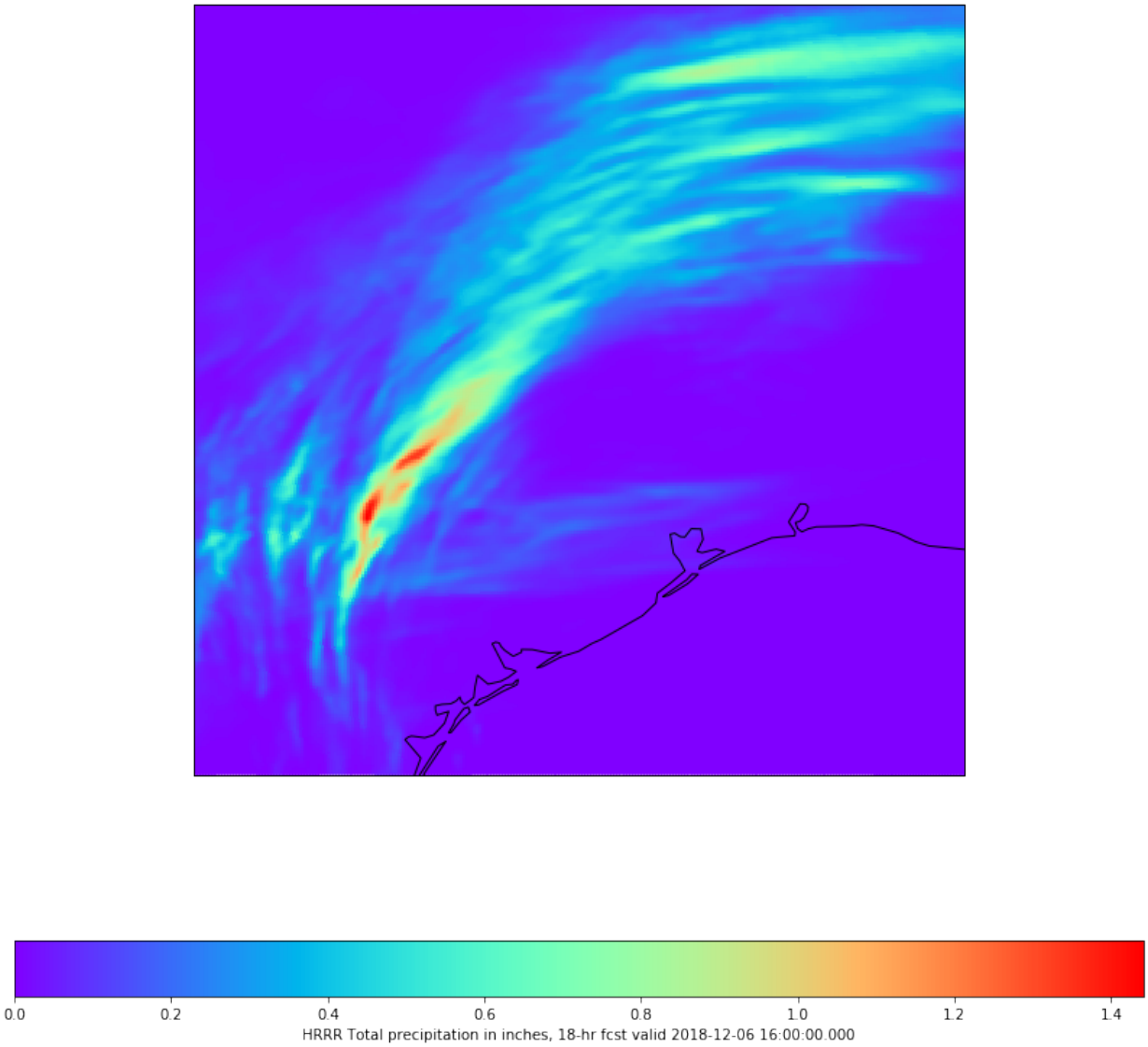
```
request = DataAccessLayer.newDataRequest("grid", envelope=polygon)
request.setLocationNames("HRRR")
request.setLevels("0.0SFC")
request.setParameters("TP")

cycles = DataAccessLayer.getAvailableTimes(request, True)
times = DataAccessLayer.getAvailableTimes(request)
fcstRun = DataAccessLayer.getForecastRun(cycles[-2], times)


for i, tt in enumerate(fcstRun):
    response = DataAccessLayer.getGridData(request, [tt])
    grid = response[0]
    if i>0:
        data += grid.getRawData()
    else:
        data = grid.getRawData()
    data[data <= -9999] = 0
    print(data.min(), data.max(), grid.getDataTime().getFcstTime()/3600)


lons, lats = grid.getLatLonCoords()
bbox = [lons.min(), lons.max(), lats.min(), lats.max()]
fcstHr = int(grid.getDataTime().getFcstTime()/3600)

tp_inch = data * (0.0393701)
print(tp_inch.min(), tp_inch.max())
```

(continues on next page)

(continued from previous page)

```
def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(20, 14),
            subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    return fig, ax

cmap = plt.get_cmap('rainbow')
fig, ax = make_map(bbox=box)
cs = ax.pcolormesh(lons, lats, tp_inch, cmap=cmap)
cbar = fig.colorbar(cs, shrink=0.7, orientation='horizontal')
cbar.set_label(grid.getLocationName() + " Total precipitation in inches, " \
               + str(fcstHr) + "-hr fcst valid " + str(grid.getDataTime().
↪getRefTime()))
```

```
0.0 1.853 1.0
0.0 3.5290003 2.0
0.0 5.0290003 3.0
0.0 5.051 4.0
0.0 5.2960005 5.0
0.0 5.2960005 6.0
0.0 5.8269997 7.0
0.0 6.1790004 8.0
0.0 6.1890006 9.0
0.0 9.071 10.0
0.0 10.812 11.0
0.0 14.718 12.0
0.0 18.295 13.0
0.0 21.339 14.0
0.0 22.626 15.0
0.0 28.670002 16.0
0.0 32.334 17.0
0.0 36.628002 18.0
0.0 1.4420482
```



Profiler Wind Barb Time-Series

Notebook

```
from awips.dataaccess import DataAccessLayer
from awips.tables import profiler
import matplotlib.tri as mtri
from datetime import datetime, timedelta
from matplotlib.dates import date2num
from metpy.units import units
import numpy as np
import six

# Query ESRL/PSD profiler data from Unidata AWIPS
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest("profiler")
profilerSites = DataAccessLayer.getAvailableLocationNames(request)
```

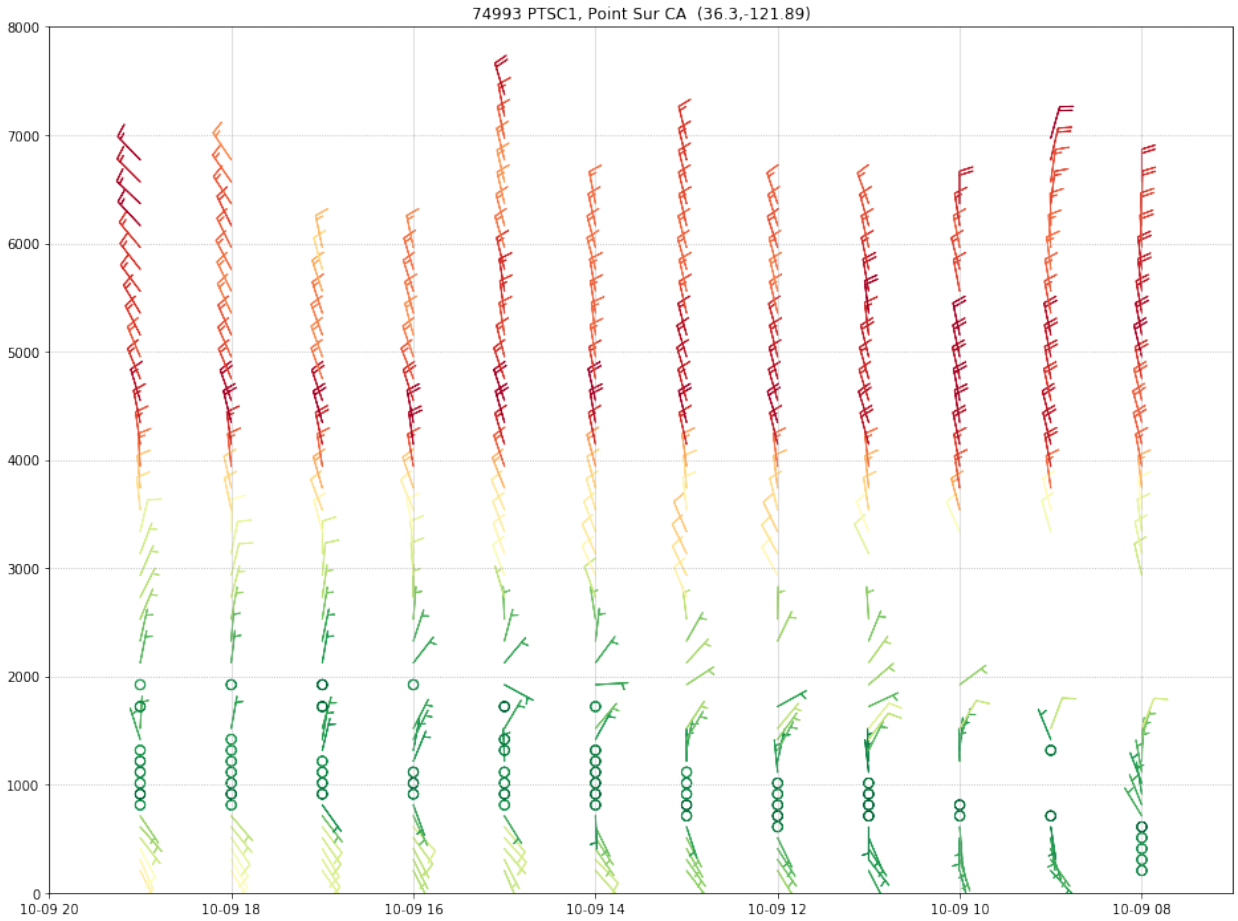
(continues on next page)

(continued from previous page)

```
print(profilerSites)
import matplotlib.pyplot as plt
```

```
['74992', '74998', '74996', '74995', '74997', '74994', '74991', '74990', '74993']
```

```
%matplotlib inline
fig = plt.figure(figsize=(16,12))
site = profilerSites[-1]

# Request the last twelve hourly obs
request = DataAccessLayer.newDataRequest("profiler")
request.setLocationNames(site)
request.setParameters("uComponent","vComponent")
hrs=12
requestTimes = DataAccessLayer.getAvailableTimes(request)[-1*hrs:]
response = DataAccessLayer.getGeometryData(request,requestTimes)

# Create a plot for each station
ax = fig.add_subplot(1,1,1)
ax.title.set_text(site +" "+ str(profiler[site]['profilerId']) \
    +", "+ str(profiler[site]['profilerName']) \
    +"  ("+ str(profiler[site]['latitude']) +","+ str(profiler[site]['longitude']) +")
↪")
ax.set_ylim(0,8000)
ax.grid(axis='x', which='major', alpha=0.5)
ax.grid(axis='y', which='major', linestyle=':')
ax.xaxis_date()
ax.set_xlim(min(requestTimes).validPeriod.start-timedelta(hours=1), max(requestTimes).
↪validPeriod.start+timedelta(hours=1))

# Plot profiler observations
for time in requestTimes:
    data,t=[],[]
    u,v=[],[]
    for ob in response:
        if str(ob.getDataTime().getValidPeriod().start) == str(time):
            data_tuple = (float(ob.getLevel().replace('FHAG','')),
                float(ob.getNumber("uComponent")),
                float(ob.getNumber("vComponent")))
            data.append(data_tuple)
            t.append(time.validPeriod.start)

    data = np.array(data, dtype=[
        ('h', np.float32),
        ('u', np.float32),
        ('v', np.float32)])

    u = data['u']
    v = data['v']
    h = data['h']
    C = np.sqrt(u**2 + v**2)
    ax.barbs(date2num(t), h, u, v, C, cmap=plt.cm.RdYlGn_r)

plt.gca().invert_xaxis()
```



Regional Surface Obs Plot

Notebook

```
%matplotlib inline
```

This exercise creates a surface observsation station plot for the state of Florida, using both METAR (datatype *obs*) and Synoptic (datatype *sfcobs*). Because we are using the AWIPS Map Database for state and county boundaries, there is no use of Cartopy `cfeature` in this exercise.

```
from awips.dataaccess import DataAccessLayer
from dynamicserialize.dstypes.com.raytheon.uf.common.time import TimeRange
from datetime import datetime, timedelta
import numpy as np
import cartopy.crs as ccrs
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
from cartopy.feature import ShapelyFeature
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
from metpy.units import units
from metpy.calc import wind_components
from metpy.plots import simple_layout, StationPlot, StationPlotLayout
import warnings
```

(continues on next page)

(continued from previous page)

```
def get_cloud_cover(code):
    if 'OVC' in code:
        return 1.0
    elif 'BKN' in code:
        return 6.0/8.0
    elif 'SCT' in code:
        return 4.0/8.0
    elif 'FEW' in code:
        return 2.0/8.0
    else:
        return 0
```

```
# EDEX request for a single state
edexServer = "edex-cloud.unidata.ucar.edu"
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest('maps')
request.addIdentifier('table', 'mapdata.states')
request.addIdentifier('state', 'FL')
request.addIdentifier('geomField', 'the_geom')
request.setParameters('state','name','lat','lon')
response = DataAccessLayer.getGeometryData(request)
record = response[0]
print("Found " + str(len(response)) + " MultiPolygon")
state={}
state['name'] = record.getString('name')
state['state'] = record.getString('state')
state['lat'] = record.getNumber('lat')
state['lon'] = record.getNumber('lon')
#state['geom'] = record.getGeometry()
state['bounds'] = record.getGeometry().bounds
print(state['name'], state['state'], state['lat'], state['lon'], state['bounds'])
print()

# EDEX request for multiple states
request = DataAccessLayer.newDataRequest('maps')
request.addIdentifier('table', 'mapdata.states')
request.addIdentifier('geomField', 'the_geom')
request.addIdentifier('inLocation', 'true')
request.addIdentifier('locationField', 'state')
request.setParameters('state','name','lat','lon')
request.setLocationNames('FL','GA','MS','AL','SC','LA')
response = DataAccessLayer.getGeometryData(request)
print("Found " + str(len(response)) + " MultiPolygons")

# Append each geometry to a numpy array
states = np.array([])
for ob in response:
    print(ob.getString('name'), ob.getString('state'), ob.getNumber('lat'), ob.
↪getNumber('lon'))
    states = np.append(states,ob.getGeometry())
```

```
Found 1 MultiPolygon
Florida FL 28.67402 -82.50934 (-87.63429260299995, 24.521051616000022, -80.
↪03199876199994, 31.001012802000048)

Found 6 MultiPolygons
```

(continues on next page)

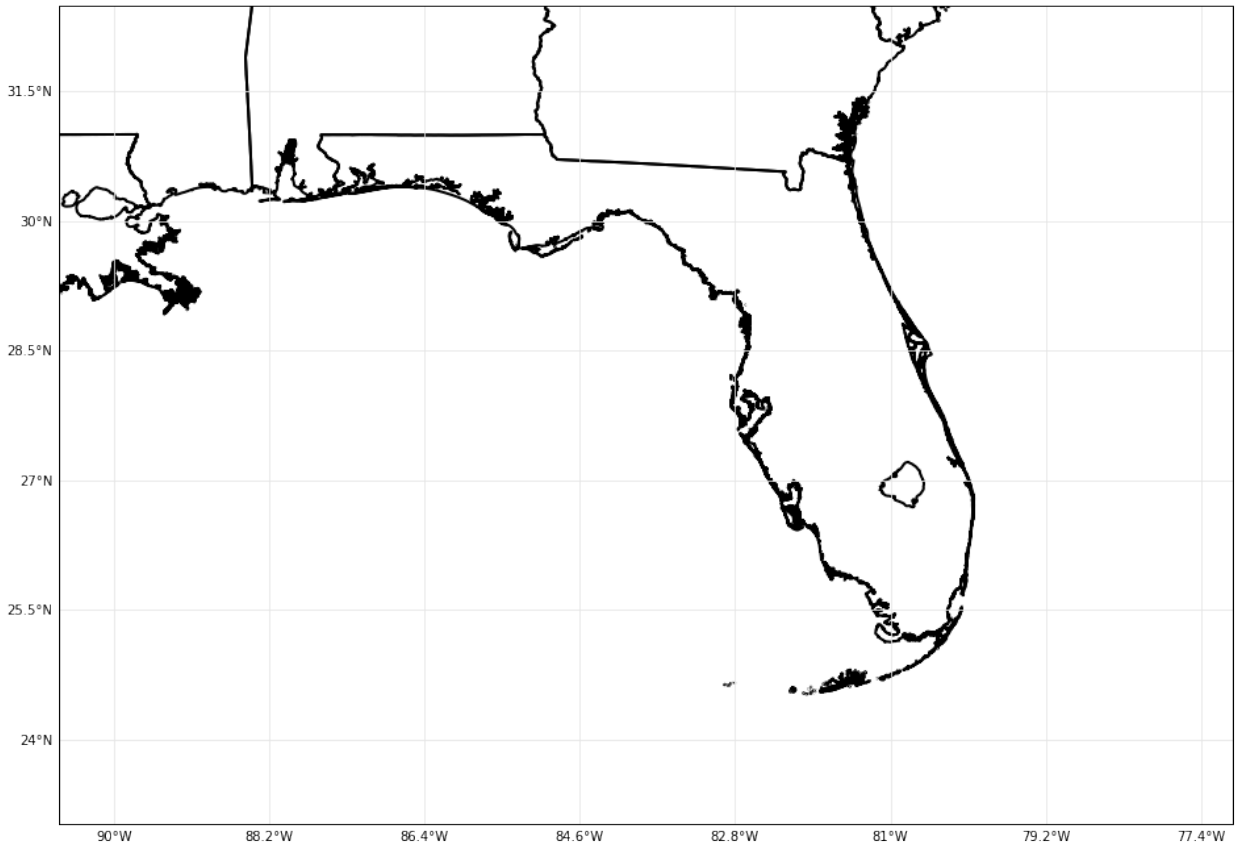
(continued from previous page)

```
Florida FL 28.67402 -82.50934
Georgia GA 32.65155 -83.44848
Louisiana LA 31.0891 -92.02905
Alabama AL 32.79354 -86.82676
Mississippi MS 32.75201 -89.66553
South Carolina SC 33.93574 -80.89899
```

Now make sure we can plot the states with a lat/lon grid.

```
def make_map(bbox, proj=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(16,12),subplot_kw=dict(projection=proj))
    ax.set_extent(bbox)
    gl = ax.gridlines(draw_labels=True, color='#e7e7e7')
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax

# buffer our bounds by +/i degrees lat/lon
bounds = state['bounds']
bbox=[bounds[0]-3,bounds[2]+3,bounds[1]-1.5,bounds[3]+1.5]

fig, ax = make_map(bbox=bbox)
shape_feature = ShapelyFeature(states,ccrs.PlateCarree(),
                    facecolor='none', linestyle="-",edgecolor='#000000',linewidth=2)
ax.add_feature(shape_feature)
```

```
<cartopy.mpl.feature_artist.FeatureArtist at 0x11dcfedd8>
```



Plot METAR (obs)

Here we use a spatial envelope to limit the request to the boundary or our plot. Without such a filter you may be requesting many tens of thousands of records.

```
# Create envelope geometry
envelope = Polygon([(bbox[0],bbox[2]),(bbox[0],bbox[3]),
                    (bbox[1], bbox[3]),(bbox[1],bbox[2]),
                    (bbox[0],bbox[2])])

# New obs request
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest("obs", envelope=envelope)
availableProducts = DataAccessLayer.getAvailableParameters(request)
single_value_params = ["timeObs", "stationName", "longitude", "latitude",
                       "temperature", "dewpoint", "windDir",
                       "windSpeed", "seaLevelPress"]
multi_value_params = ["presWeather", "skyCover", "skyLayerBase"]
params = single_value_params + multi_value_params
request.setParameters(*(params))

# Time range
lastHourDateTime = datetime.utcnow() - timedelta(minutes = 60)
start = lastHourDateTime.strftime('%Y-%m-%d %H:%M:%S')
```

(continues on next page)

(continued from previous page)

```
end = datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')

beginRange = datetime.strptime( start , "%Y-%m-%d %H:%M:%S")
endRange = datetime.strptime( end , "%Y-%m-%d %H:%M:%S")
timerange = TimeRange(beginRange, endRange)
# Get response
response = DataAccessLayer.getGeometryData(request,timerange)
# function getMetarObs was added in python-awips 18.1.4
obs = DataAccessLayer.getMetarObs(response)
print("Found " + str(len(response)) + " records")
print("Using " + str(len(obs['temperature'])) + " temperature records")
```

```
Found 3468 records
Using 152 temperature records
```

Next grab the simple variables out of the data we have (attaching correct units), and put them into a dictionary that we will hand the plotting function later:

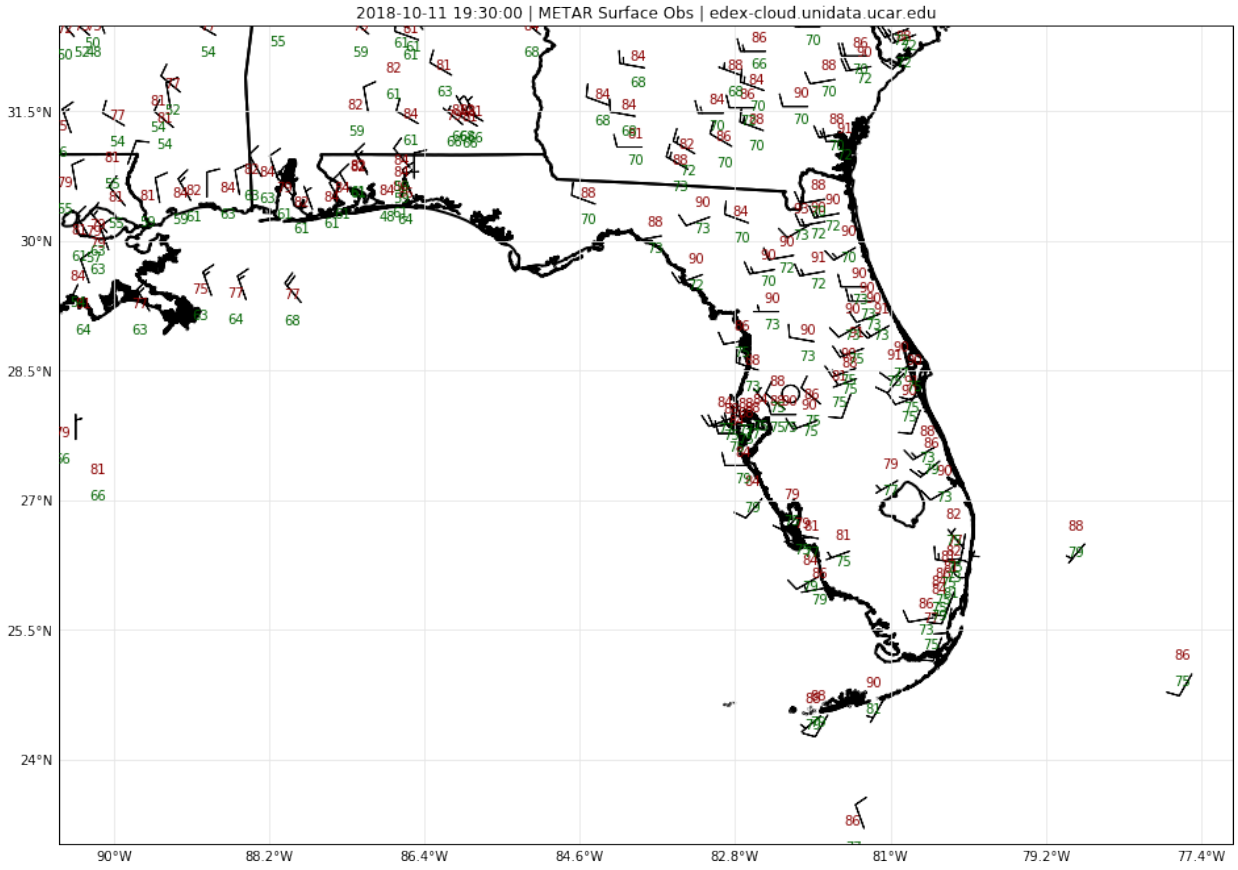
- Get wind components from speed and direction
- Convert cloud fraction values to integer codes [0 - 8]
- Map METAR weather codes to WMO codes for weather symbols

```
data = dict()
data['stid'] = np.array(obs['stationName'])
data['latitude']  = np.array(obs['latitude'])
data['longitude'] = np.array(obs['longitude'])
tmp = np.array(obs['temperature'], dtype=float)
dpt = np.array(obs['dewpoint'], dtype=float)

# Suppress nan masking warnings
warnings.filterwarnings("ignore",category =RuntimeWarning)

tmp[tmp == -9999.0] = 'nan'
dpt[dpt == -9999.] = 'nan'
data['air_temperature'] = tmp * units.degC
data['dew_point_temperature'] = dpt * units.degC
data['air_pressure_at_sea_level'] = np.array(obs['seaLevelPress'])* units('mbar')
direction = np.array(obs['windDir'])
direction[direction == -9999.0] = 'nan'
u, v = wind_components(np.array(obs['windSpeed']) * units('knots'),
                       direction * units.degree)
data['eastward_wind'], data['northward_wind'] = u, v
data['cloud_coverage'] = [int(get_cloud_cover(x)*8) for x in obs['skyCover']]
data['present_weather'] = obs['presWeather']
proj = ccrs.LambertConformal(central_longitude=state['lon'], central_latitude=state[
↪'lat'],
                             standard_parallels=[35])
custom_layout = StationPlotLayout()
custom_layout.add_barb('eastward_wind', 'northward_wind', units='knots')
custom_layout.add_value('NW', 'air_temperature', fmt='.0f', units='degF', color=
↪'darkred')
custom_layout.add_value('SW', 'dew_point_temperature', fmt='.0f', units='degF', color=
↪'darkgreen')
custom_layout.add_value('E', 'precipitation', fmt='0.1f', units='inch', color='blue')
ax.set_title(str(response[-1].getDataTime()) + " | METAR Surface Obs | " + edexServer)
```

(continues on next page)

(continued from previous page)

```
stationplot = StationPlot(ax, data['longitude'], data['latitude'], clip_on=True,
                          transform=ccrs.PlateCarree(), fontsize=10)
custom_layout.plot(stationplot, data)
fig
```



Plot Synoptic (sfcobs)

```
# New sfcobs/SYNOP request
DataAccessLayer.changeEDEXHost(edexServer)
request = DataAccessLayer.newDataRequest("sfcobs", envelope=envelope)
availableProducts = DataAccessLayer.getAvailableParameters(request)
# (sfcobs) uses stationId, while (obs) uses stationName,
# the rest of these parameters are the same.
single_value_params = ["timeObs", "stationId", "longitude", "latitude",
                       "temperature", "dewpoint", "windDir",
                       "windSpeed", "seaLevelPress"]
multi_value_params = ["presWeather", "skyCover", "skyLayerBase"]
pres_weather, sky_cov, sky_layer_base = [],[],[]
params = single_value_params + multi_value_params
request.setParameters(*(params))

# Time range
```

(continues on next page)

(continued from previous page)

```
lastHourDateTime = datetime.utcnow() - timedelta(minutes = 60)
start = lastHourDateTime.strftime('%Y-%m-%d %H:%M:%S')
end = datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')

beginRange = datetime.strptime( start , "%Y-%m-%d %H:%M:%S")
endRange = datetime.strptime( end , "%Y-%m-%d %H:%M:%S")
timerange = TimeRange(beginRange, endRange)

# Get response
response = DataAccessLayer.getGeometryData(request,timerange)
# function getSynopticObs was added in python-awips 18.1.4
sfcobs = DataAccessLayer.getSynopticObs(response)
print("Found " + str(len(response)) + " records")
print("Using " + str(len(sfcobs['temperature'])) + " temperature records")
```

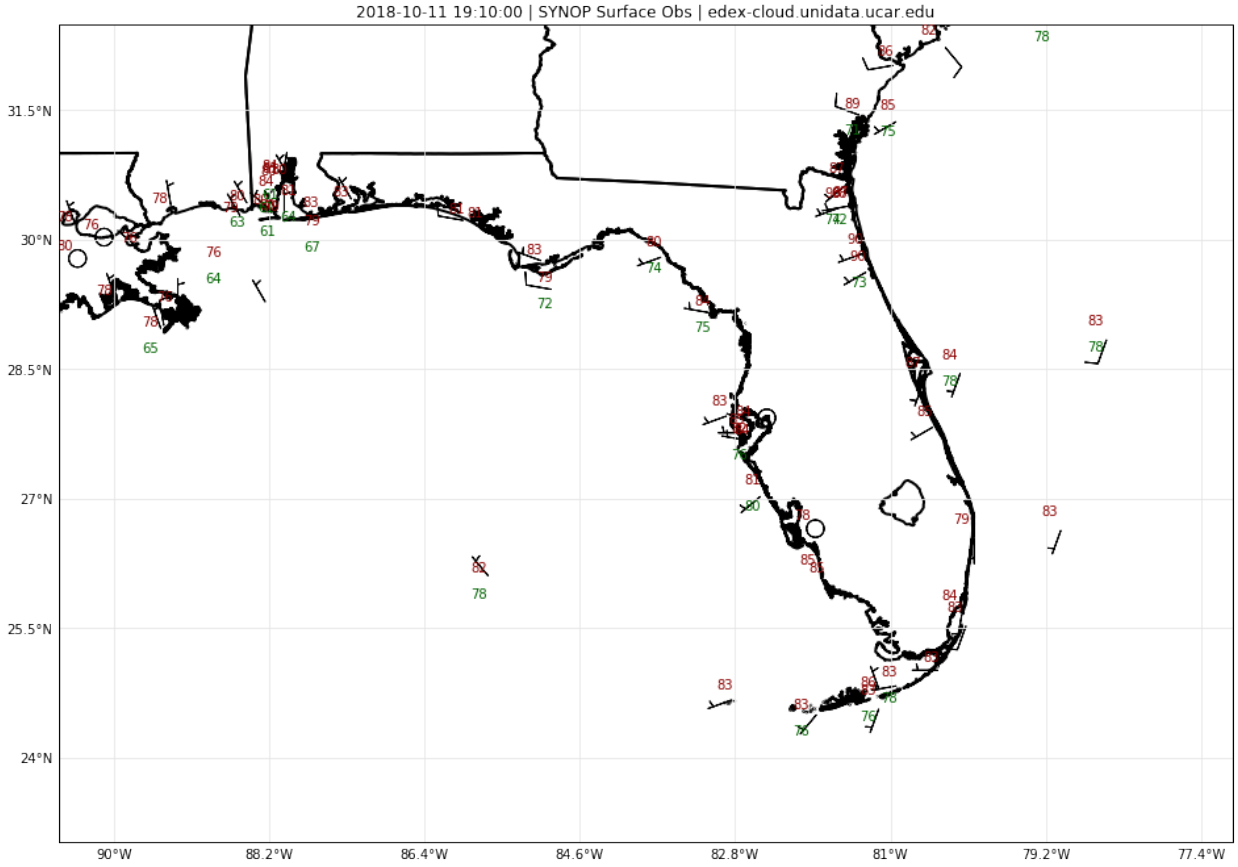
```
Found 260 records
Using 78 temperature records
```

```
data = dict()
data['stid'] = np.array(sfcobs['stationId'])
data['lat']  = np.array(sfcobs['latitude'])
data['lon'] = np.array(sfcobs['longitude'])

# Synop/sfcobs temps are stored in kelvin (degC for METAR/obs)
tmp = np.array(sfcobs['temperature'], dtype=float)
dpt = np.array(sfcobs['dewpoint'], dtype=float)
direction = np.array(sfcobs['windDir'])
# Account for missing values
tmp[tmp == -9999.0] = 'nan'
dpt[dpt == -9999.] = 'nan'
direction[direction == -9999.0] = 'nan'

data['air_temperature'] = tmp * units.kelvin
data['dew_point_temperature'] = dpt * units.kelvin
data['air_pressure_at_sea_level'] = np.array(sfcobs['seaLevelPress'])* units('mbar')
try:
    data['eastward_wind'], data['northward_wind'] = wind_components(
        np.array(sfcobs['windSpeed']) * units('knots'),direction * units.degree)
    data['present_weather'] = sfcobs['presWeather']
except ValueError:
    pass

fig_synop, ax_synop = make_map(bbox=bbox)
shape_feature = ShapelyFeature(states,ccrs.PlateCarree(),
                    facecolor='none', linestyle="-",edgecolor='#000000',linewidth=2)
ax_synop.add_feature(shape_feature)

custom_layout = StationPlotLayout()
custom_layout.add_barb('eastward_wind', 'northward_wind', units='knots')
custom_layout.add_value('NW', 'air_temperature', fmt='.0f', units='degF', color=
↪'darkred')
custom_layout.add_value('SW', 'dew_point_temperature', fmt='.0f', units='degF', color=
↪'darkgreen')
custom_layout.add_value('E', 'precipitation', fmt='0.1f', units='inch', color='blue')
ax_synop.set_title(str(response[-1].getDataTime()) + " | SYNOP Surface Obs | " +␣
↪edexServer)
```

(continues on next page)

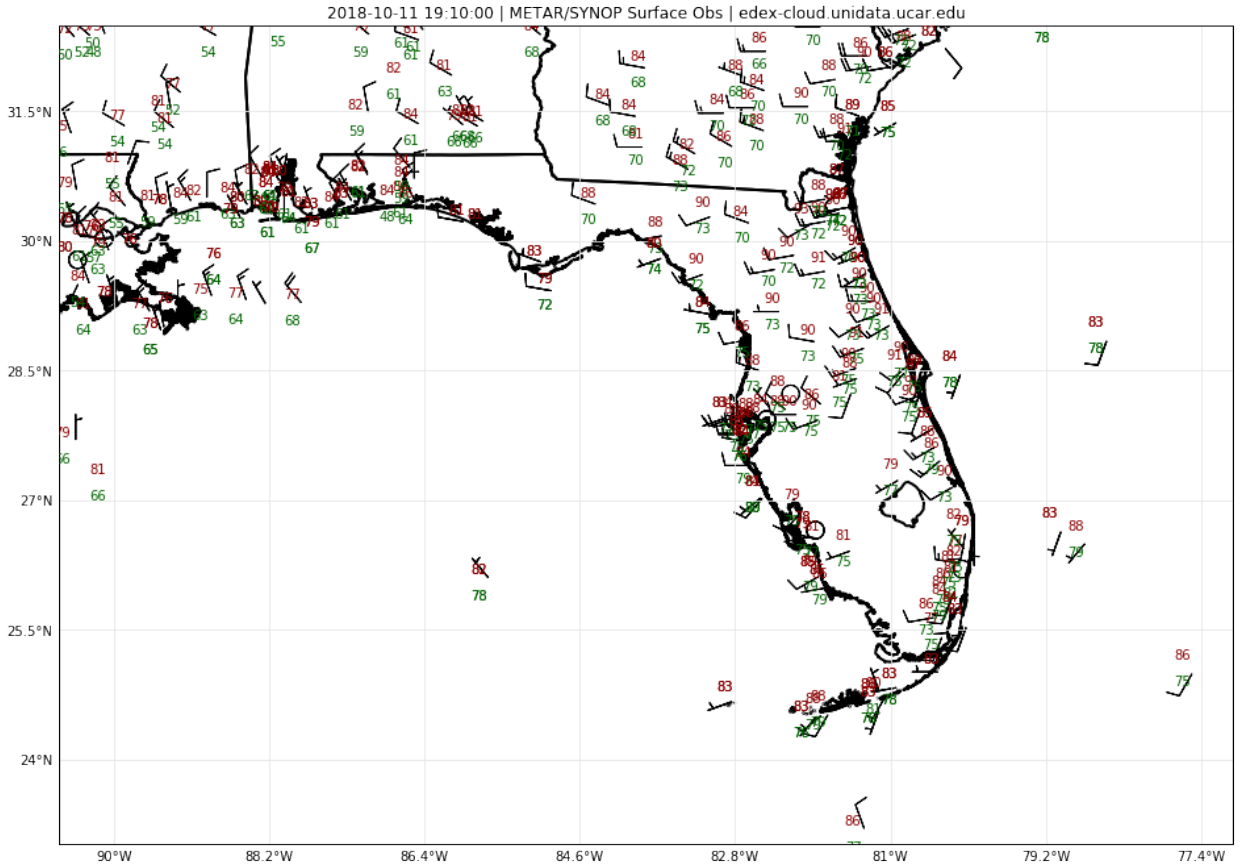
(continued from previous page)

```
stationplot = StationPlot(ax_synop, data['lon'], data['lat'], clip_on=True,
                          transform=ccrs.PlateCarree(), fontsize=10)
custom_layout.plot(stationplot, data)
```



Plot both METAR and SYNOP

```
custom_layout = StationPlotLayout()
custom_layout.add_barb('eastward_wind', 'northward_wind', units='knots')
custom_layout.add_value('NW', 'air_temperature', fmt='.0f', units='degF', color=
↪'darkred')
custom_layout.add_value('SW', 'dew_point_temperature', fmt='.0f', units='degF', color=
↪'darkgreen')
custom_layout.add_value('E', 'precipitation', fmt='0.1f', units='inch', color='blue')
ax.set_title(str(response[-1].getDataTime()) + " | METAR/SYNOP Surface Obs | " +␣
↪edexServer)
stationplot = StationPlot(ax, data['lon'], data['lat'], clip_on=True,
                          transform=ccrs.PlateCarree(), fontsize=10)
custom_layout.plot(stationplot, data)
fig
```



Satellite Imagery

Notebook

Satellite images are returned by Python AWIPS as grids, and can be rendered with Cartopy pcolormesh the same as gridded forecast models in other python-awips examples.

Available Sources, Creating Entities, Sectors, and Products

```
from awips.dataaccess import DataAccessLayer
import cartopy.crs as ccrs
import cartopy.feature as cfeat
import matplotlib.pyplot as plt
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
import numpy as np
import datetime

# Create an EDEX data request
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest()
request.setDatatype("satellite")

# get optional identifiers for satellite datatype
identifiers = set(DataAccessLayer.getOptionalIdentifiers(request))
```

(continues on next page)

(continued from previous page)

```
print("Available Identifiers:")
for id in identifiers:
    if id.lower() == 'datauri':
        continue
    print(" - " + id)
```

```
Available Identifiers:
 - physicalElement
 - creatingEntity
 - source
 - sectorID
```

```
# Show available sources
identifier = "source"
sources = DataAccessLayer.getIdentifierValues(request, identifier)
print(identifier + ":")
print(list(sources))
```

```
source:
['NESDIS', 'WCDAS', 'NSOF', 'UCAR', 'McIDAS']
```

```
# Show available creatingEntities
identifier = "creatingEntity"
creatingEntities = DataAccessLayer.getIdentifierValues(request, identifier)
print(identifier + ":")
print(list(creatingEntities))
```

```
creatingEntity:
['GOES-16', 'Composite', 'GOES-15(P)', 'POES-NPOESS', 'UNIWISC', 'GOES-11(L)',
↪'Miscellaneous', 'GOES-17', 'NEXRCOMP']
```

```
# Show available sectorIDs
identifier = "sectorID"
sectorIDs = DataAccessLayer.getIdentifierValues(request, identifier)
print(identifier + ":")
print(list(sectorIDs))
```

```
sectorID:
['EMESO-2', 'Northern Hemisphere Composite', 'EFD', 'TCONUS', 'Arctic', 'TFD', 'PRREGI
↪', 'GOES-Sounder', 'EMESO-1', 'NEXRCOMP', 'ECONUS', 'GOES-West', 'Antarctic', 'GOES-
↪East', 'Supernational', 'West CONUS', 'NH Composite - Meteosat-GOES E-GOES W-GMS']
```

```
# Contrust a full satellite product tree
for entity in creatingEntities:
    print(entity)
    request = DataAccessLayer.newDataRequest("satellite")
    request.addIdentifier("creatingEntity", entity)
    availableSectors = DataAccessLayer.getAvailableLocationNames(request)
    availableSectors.sort()
    for sector in availableSectors:
        print(" - " + sector)
        request.setLocationNames(sector)
        availableProducts = DataAccessLayer.getAvailableParameters(request)
```

(continues on next page)

(continued from previous page)

```
        availableProducts.sort()
        for product in availableProducts:
            print("    - " + product)
```

```
GOES-16
 - ECONUS
    - ACTP
    - ADP
    - AOD
    - CAPE
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
    - CSM
    - CTH
    - FDC Area
    - FDC Power
    - FDC Temp
    - KI
    - LI
    - LST
    - SI
    - TPW
    - TT
    - VMP-0.00hPa
    - VMP-0.02hPa
    - VMP-0.04hPa
    - VMP-0.08hPa
    - VMP-0.14hPa
    - VMP-0.22hPa
    - VMP-0.35hPa
    - VMP-0.51hPa
    - VMP-0.71hPa
    - VMP-0.98hPa
    - VMP-1.30hPa
    - VMP-1.69hPa
    - VMP-1013.95hPa
    - VMP-103.02hPa
    - VMP-1042.23hPa
    - VMP-1070.92hPa
    - VMP-11.00hPa
    - VMP-110.24hPa
    - VMP-1100.00hPa
```

(continues on next page)

(continued from previous page)

```
    - VMP-117.78hPa
    - VMP-12.65hPa
    - VMP-125.65hPa
    - VMP-133.85hPa
    - VMP-14.46hPa
    - VMP-142.38hPa
    - VMP-151.27hPa
    - VMP-16.43hPa
    - VMP-160.50hPa
    - VMP-170.08hPa
    - VMP-18.58hPa
    - VMP-180.02hPa
    - VMP-190.32hPa
    - VMP-2.15hPa
    - VMP-2.70hPa
    - VMP-20.92hPa
    - VMP-200.99hPa
    - VMP-212.03hPa
    - VMP-223.44hPa
    - VMP-23.45hPa
    - VMP-235.23hPa
    - VMP-247.41hPa
    - VMP-259.97hPa
    - VMP-26.18hPa
    - VMP-272.92hPa
    - VMP-286.26hPa
    - VMP-29.12hPa
    - VMP-3.34hPa
    - VMP-300.00hPa
    - VMP-314.14hPa
    - VMP-32.27hPa
    - VMP-328.68hPa
    - VMP-343.62hPa
    - VMP-35.65hPa
    - VMP-358.97hPa
    - VMP-374.72hPa
    - VMP-39.26hPa
    - VMP-390.89hPa
    - VMP-4.08hPa
    - VMP-4.92hPa
    - VMP-407.47hPa
    - VMP-424.47hPa
    - VMP-43.10hPa
    - VMP-441.88hPa
    - VMP-459.71hPa
    - VMP-47.19hPa
    - VMP-477.96hPa
    - VMP-496.63hPa
    - VMP-5.88hPa
    - VMP-51.53hPa
    - VMP-515.72hPa
    - VMP-535.23hPa
    - VMP-555.17hPa
    - VMP-56.13hPa
    - VMP-575.52hPa
    - VMP-596.31hPa
    - VMP-6.96hPa
```

(continues on next page)

(continued from previous page)

```
    - VMP-60.99hPa
    - VMP-617.51hPa
    - VMP-639.14hPa
    - VMP-66.13hPa
    - VMP-661.19hPa
    - VMP-683.67hPa
    - VMP-706.57hPa
    - VMP-71.54hPa
    - VMP-729.89hPa
    - VMP-753.63hPa
    - VMP-77.24hPa
    - VMP-777.79hPa
    - VMP-8.17hPa
    - VMP-802.37hPa
    - VMP-827.37hPa
    - VMP-83.23hPa
    - VMP-852.79hPa
    - VMP-878.62hPa
    - VMP-89.52hPa
    - VMP-9.51hPa
    - VMP-904.87hPa
    - VMP-931.52hPa
    - VMP-958.59hPa
    - VMP-96.11hPa
    - VMP-986.07hPa
    - VTP-0.00hPa
    - VTP-0.02hPa
    - VTP-0.04hPa
    - VTP-0.08hPa
    - VTP-0.14hPa
    - VTP-0.22hPa
    - VTP-0.35hPa
    - VTP-0.51hPa
    - VTP-0.71hPa
    - VTP-0.98hPa
    - VTP-1.30hPa
    - VTP-1.69hPa
    - VTP-1013.95hPa
    - VTP-103.02hPa
    - VTP-1042.23hPa
    - VTP-1070.92hPa
    - VTP-11.00hPa
    - VTP-110.24hPa
    - VTP-1100.00hPa
    - VTP-117.78hPa
    - VTP-12.65hPa
    - VTP-125.65hPa
    - VTP-133.85hPa
    - VTP-14.46hPa
    - VTP-142.38hPa
    - VTP-151.27hPa
    - VTP-16.43hPa
    - VTP-160.50hPa
    - VTP-170.08hPa
    - VTP-18.58hPa
    - VTP-180.02hPa
    - VTP-190.32hPa
```

(continues on next page)

(continued from previous page)

```
    - VTP-2.15hPa
    - VTP-2.70hPa
    - VTP-20.92hPa
    - VTP-200.99hPa
    - VTP-212.03hPa
    - VTP-223.44hPa
    - VTP-23.45hPa
    - VTP-235.23hPa
    - VTP-247.41hPa
    - VTP-259.97hPa
    - VTP-26.18hPa
    - VTP-272.92hPa
    - VTP-286.26hPa
    - VTP-29.12hPa
    - VTP-3.34hPa
    - VTP-300.00hPa
    - VTP-314.14hPa
    - VTP-32.27hPa
    - VTP-328.68hPa
    - VTP-343.62hPa
    - VTP-35.65hPa
    - VTP-358.97hPa
    - VTP-374.72hPa
    - VTP-39.26hPa
    - VTP-390.89hPa
    - VTP-4.08hPa
    - VTP-4.92hPa
    - VTP-407.47hPa
    - VTP-424.47hPa
    - VTP-43.10hPa
    - VTP-441.88hPa
    - VTP-459.71hPa
    - VTP-47.19hPa
    - VTP-477.96hPa
    - VTP-496.63hPa
    - VTP-5.88hPa
    - VTP-51.53hPa
    - VTP-515.72hPa
    - VTP-535.23hPa
    - VTP-555.17hPa
    - VTP-56.13hPa
    - VTP-575.52hPa
    - VTP-596.31hPa
    - VTP-6.96hPa
    - VTP-60.99hPa
    - VTP-617.51hPa
    - VTP-639.14hPa
    - VTP-66.13hPa
    - VTP-661.19hPa
    - VTP-683.67hPa
    - VTP-706.57hPa
    - VTP-71.54hPa
    - VTP-729.89hPa
    - VTP-753.63hPa
    - VTP-77.24hPa
    - VTP-777.79hPa
    - VTP-8.17hPa
```

(continues on next page)

(continued from previous page)

```
    - VTP-802.37hPa
    - VTP-827.37hPa
    - VTP-83.23hPa
    - VTP-852.79hPa
    - VTP-878.62hPa
    - VTP-89.52hPa
    - VTP-9.51hPa
    - VTP-904.87hPa
    - VTP-931.52hPa
    - VTP-958.59hPa
    - VTP-96.11hPa
    - VTP-986.07hPa
- EFD
    - ACTP
    - ADP
    - AOD
    - CAPE
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
    - CSM
    - CTH
    - CTT
    - FDC Area
    - FDC Power
    - FDC Temp
    - KI
    - LI
    - LST
    - RRQPE
    - SI
    - SST
    - TPW
    - TT
    - VAH
    - VAML
- EMESO-1
    - ACTP
    - ADP
    - CAPE
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
```

(continues on next page)

(continued from previous page)

```
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
    - CSM
    - CTH
    - CTT
    - KI
    - LI
    - LST
    - SI
    - TPW
    - TT
- EMESO-2
    - ACTP
    - ADP
    - CAPE
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
    - CSM
    - CTH
    - CTT
    - KI
    - LI
    - LST
    - SI
    - TPW
    - TT
- PRREGI
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
```

(continues on next page)

(continued from previous page)

```
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
Composite
 - NH Composite - Meteosat-GOES E-GOES W-GMS
    - Imager 11 micron IR
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - Supernational
    - Gridded Cloud Amount
    - Gridded Cloud Top Pressure or Height
    - Sounder Based Derived Lifted Index (LI)
    - Sounder Based Derived Precipitable Water (PW)
    - Sounder Based Derived Surface Skin Temp (SFC Skin)
GOES-15(P)
 - Northern Hemisphere Composite
    - Imager 11 micron IR
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - Supernational
    - Imager 11 micron IR
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - West CONUS
    - Imager 11 micron IR
    - Imager 13 micron IR
    - Imager 3.9 micron IR
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
    - Sounder 11.03 micron imagery
    - Sounder 14.06 micron imagery
    - Sounder 3.98 micron imagery
    - Sounder 4.45 micron imagery
    - Sounder 6.51 micron imagery
    - Sounder 7.02 micron imagery
    - Sounder 7.43 micron imagery
    - Sounder Visible imagery
POES-NPOESS
 - Supernational
    - Rain fall rate
UNIWISC
 - Antarctic
    - Imager 11 micron IR
    - Imager 12 micron IR
    - Imager 3.5-4.0 micron IR (Fog)
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - Arctic
    - Imager 11 micron IR
    - Imager 12 micron IR
```

(continues on next page)

(continued from previous page)

```
    - Imager 3.5-4.0 micron IR (Fog)
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - GOES-East
    - Imager 11 micron IR
    - Imager 13 micron IR
    - Imager 3.5-4.0 micron IR (Fog)
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
 - GOES-Sounder
    - CAPE
    - Sounder Based Derived Lifted Index (LI)
    - Sounder Based Derived Precipitable Water (PW)
    - Sounder Based Total Column Ozone
 - GOES-West
    - Imager 11 micron IR
    - Imager 13 micron IR
    - Imager 3.5-4.0 micron IR (Fog)
    - Imager 6.7-6.5 micron IR (WV)
    - Imager Visible
GOES-11(L)
 - West CONUS
    - Low cloud base imagery
Miscellaneous
 - Supernational
    - Percent of Normal TPW
    - Sounder Based Derived Precipitable Water (PW)
GOES-17
 - TCONUS
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
 - TFD
    - CH-01-0.47um
    - CH-02-0.64um
    - CH-03-0.87um
    - CH-04-1.38um
    - CH-05-1.61um
    - CH-06-2.25um
    - CH-07-3.90um
    - CH-08-6.19um
    - CH-09-6.95um
    - CH-10-7.34um
    - CH-11-8.50um
```

(continues on next page)

(continued from previous page)

```
    - CH-12-9.61um
    - CH-13-10.35um
    - CH-14-11.20um
    - CH-15-12.30um
    - CH-16-13.30um
NEXRCOMP
 - NEXRCOMP
    - DHR
    - DVL
    - EET
    - HHC
    - N0R
    - N1P
    - NTP
```

GOES 16 Mesoscale Sectors

Define our imports, and define our map properties first.

```
%matplotlib inline

def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(10,12),
            subplot_kw=dict(projection=projection))
    if bbox[0] is not np.nan:
        ax.set_extent(bbox)
    ax.coastlines(resolution='50m')
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax

sectors = ["EMESO-1","EMESO-2"]
fig = plt.figure(figsize=(16,7*len(sectors)))

for i, sector in enumerate(sectors):

    request = DataAccessLayer.newDataRequest()
    request.setDatatype("satellite")
    request.setLocationNames(sector)
    request.setParameters("CH-13-10.35um")

    utc = datetime.datetime.utcnow()
    times = DataAccessLayer.getAvailableTimes(request)
    hourdiff = utc - datetime.datetime.strptime(str(times[-1]),'%Y-%m-%d %H:%M:%S')
    hours,days = hourdiff.seconds/3600,hourdiff.days
    minute = str((hourdiff.seconds - (3600 * hours)) / 60)
    offsetStr = ''
    if hours > 0:
        offsetStr += str(hours) + "hr "
    offsetStr += str(minute) + "m ago"
    if days > 1:
        offsetStr = str(days) + " days ago"
```

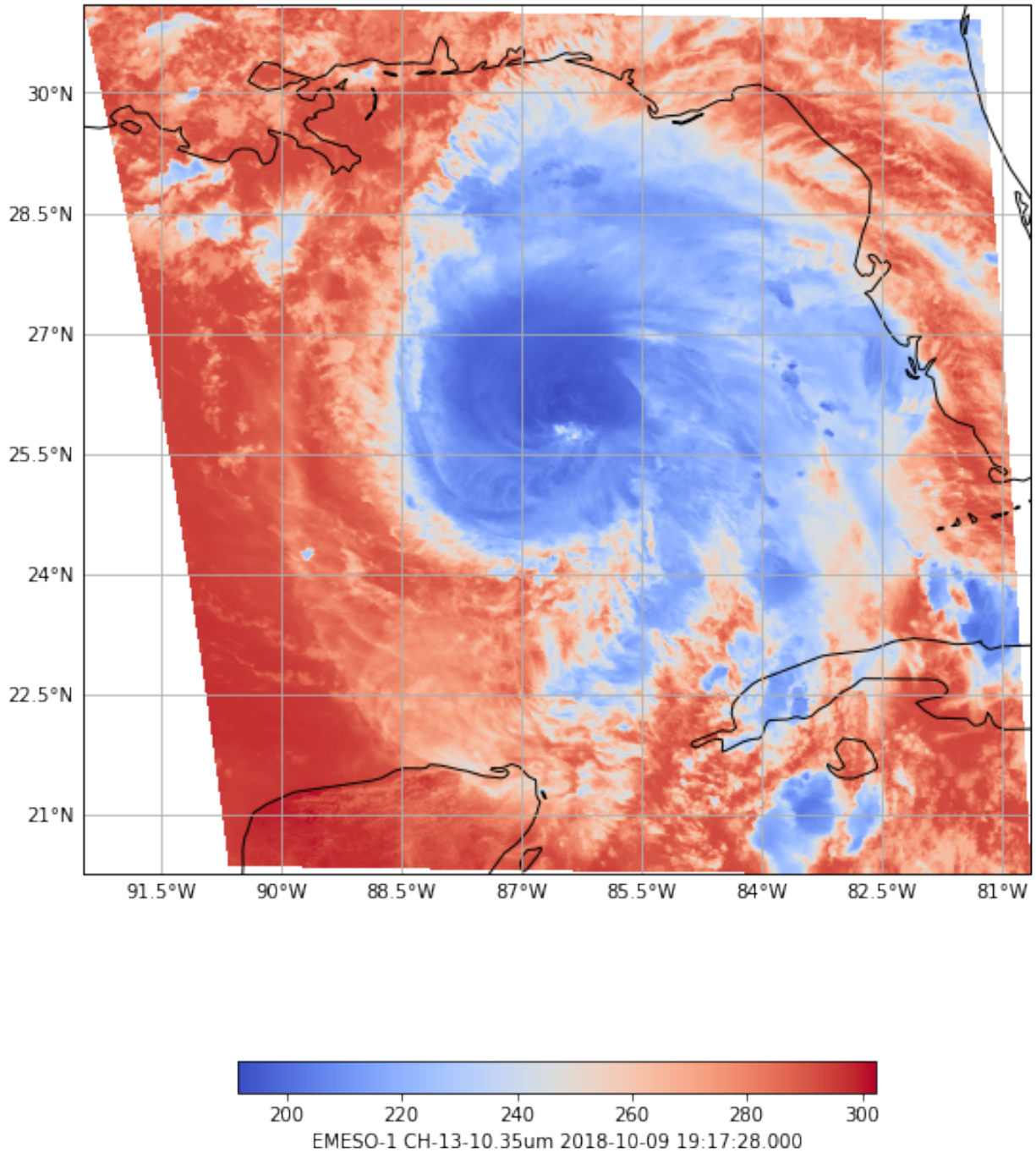
(continues on next page)

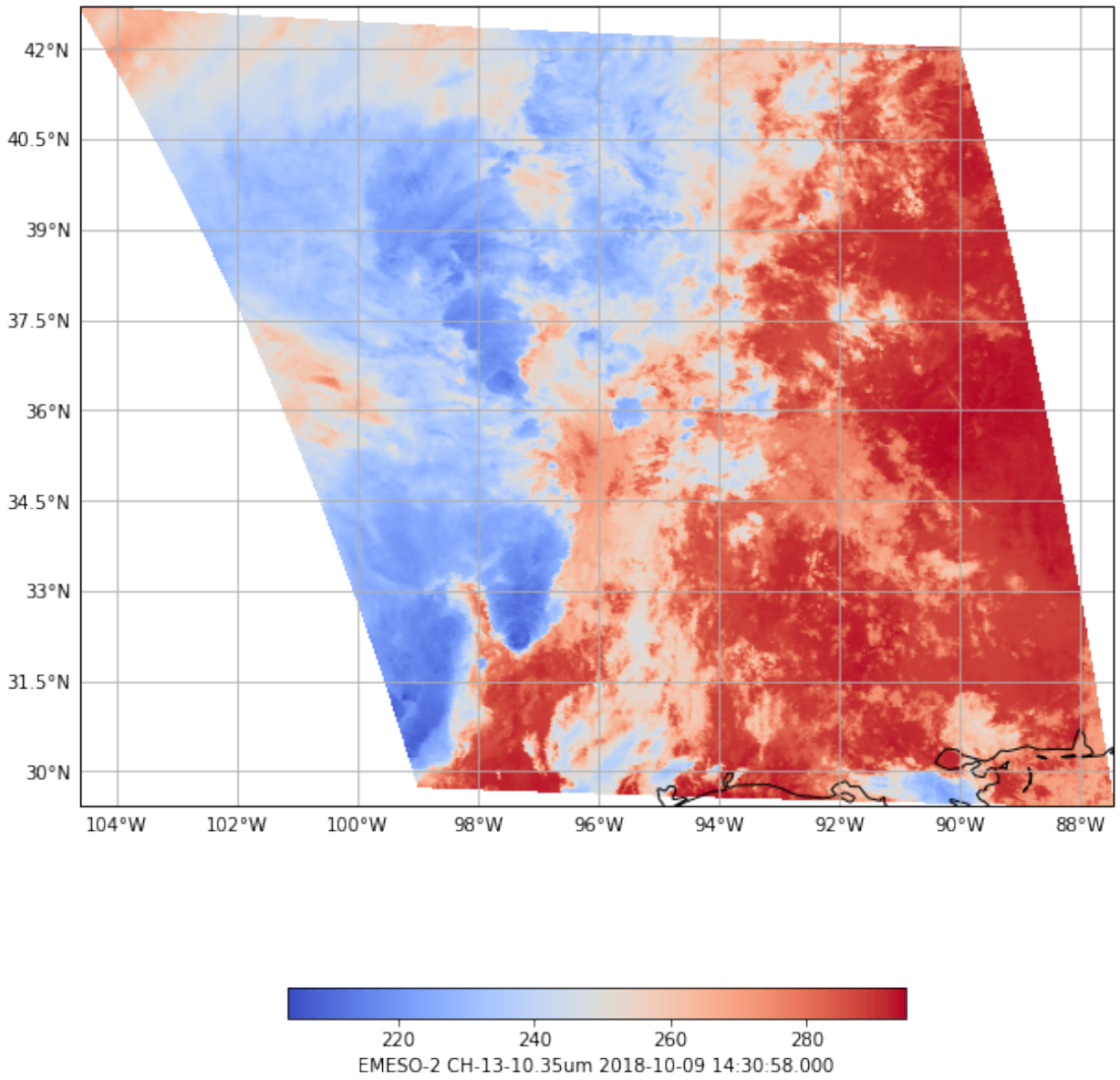
(continued from previous page)

```
    response = DataAccessLayer.getGridData(request, [times[-1]])
    grid = response[0]
    data = grid.getRawData()
    lons,lats = grid.getLatLonCoords()
    bbox = [lons.min(), lons.max(), lats.min(), lats.max()]

    print("Latest image available: "+str(times[-1]) + " ("+offsetStr+")")
    print("Image grid size: " + str(data.shape))
    print("Image grid extent: " + str(list(bbox)))

    fig, ax = make_map(bbox=bbox)
    states = cfeat.NaturalEarthFeature(category='cultural',
                                       name='admin_1_states_provinces_lines',
                                       scale='50m', facecolor='none')
    ax.add_feature(states, linestyle=':')
    cs = ax.pcolormesh(lons, lats, data, cmap='coolwarm')
    cbar = fig.colorbar(cs, shrink=0.6, orientation='horizontal')
    cbar.set_label(sector + " " + grid.getParameter() + " " \
    + str(grid.getDataTime().getRefTime()))
```

```
Latest image available: 2018-10-09 19:17:28 (0.021388888888888888hr 0.0m ago)
Image grid size: (500, 500)
Image grid extent: [-92.47462, -80.657455, 20.24799, 31.116167]
Latest image available: 2018-10-09 14:30:58 (4.797777777777778hr 0.0m ago)
Image grid size: (500, 500)
Image grid extent: [-104.61595, -87.45227, 29.422266, 42.70851]
```

```
<Figure size 1152x1008 with 0 Axes>
```


30°N
28.5°N
27°N
25.5°N
24°N
22.5°N
21°N
91.5°W
90°W
88.5°W
87°W
85.5°W
84°W
82.5°W
81°W
200
220
240
260
280
300
EMESO-1 CH-13-10.35um 2018-10-09 19:17:28.000



Upper Air BUFR Soundings

Notebook

The following script takes you through the steps of retrieving an Upper Air vertical profile from an AWIPS EDEX server and plotting a Skew-T/Log-P chart with Matplotlib and MetPy.

The **bufrua** plugin returns separate objects for parameters at **mandatory levels** and at **significant temperature levels**. For the Skew-T/Log-P plot, significant temperature levels are used to plot the pressure, temperature, and dewpoint lines, while mandatory levels are used to plot the wind profile.

```
%matplotlib inline
from awips.dataaccess import DataAccessLayer
import matplotlib.tri as mtri
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import numpy as np
import math
from metpy.calc import wind_speed, wind_components, lcl, dry_lapse, parcel_profile
from metpy.plots import SkewT, Hodograph
from metpy.units import units, concatenate

# Set host
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest()

# Set data type
request.setDatatype("bufrua")
availableLocs = DataAccessLayer.getAvailableLocationNames(request)
availableLocs.sort()

MAN_PARAMS = set(['prMan', 'htMan', 'tpMan', 'tdMan', 'wdMan', 'wsMan'])
SIGT_PARAMS = set(['prSigT', 'tpSigT', 'tdSigT'])
request.setParameters("wmoStaNum", "validTime", "rptType", "staElev", "numMand",
                      "numSigT", "numSigW", "numTrop", "numMwnd", "staName")
request.getParameters().extend(MAN_PARAMS)
request.getParameters().extend(SIGT_PARAMS)
locations = DataAccessLayer.getAvailableLocationNames(request)
locations.sort()

# Set station ID (not name)
request.setLocationNames("72562") #KLBF

# Get all times
datatimes = DataAccessLayer.getAvailableTimes(request)

# Get most recent record
response = DataAccessLayer.getGeometryData(request,times=datatimes[-1].validPeriod)

# Initialize data arrays
tdMan,tpMan,prMan,wdMan,wsMan = np.array([]),np.array([]),np.array([]),np.array([]),
↪np.array([])
prSig,tpSig,tdSig = np.array([]),np.array([]),np.array([])
manGeos = []
sigtGeos = []

# Build  arrays
for ob in response:
    parm_array = ob.getParameters()
    if set(parm_array) & MAN_PARAMS:
        manGeos.append(ob)
        prMan = np.append(prMan,ob.getNumber("prMan"))
        tpMan, tpUnit = np.append(tpMan,ob.getNumber("tpMan")), ob.getUnit("tpMan")
        tdMan, tdUnit = np.append(tdMan,ob.getNumber("tdMan")), ob.getUnit("tdMan")
        wdMan = np.append(wdMan,ob.getNumber("wdMan"))
        wsMan, wsUnit = np.append(wsMan,ob.getNumber("wsMan")), ob.getUnit("wsMan")
        continue
    if set(parm_array) & SIGT_PARAMS:
        sigtGeos.append(ob)
        prSig = np.append(prSig,ob.getNumber("prSigT"))
        tpSig = np.append(tpSig,ob.getNumber("tpSigT"))
        tdSig = np.append(tdSig,ob.getNumber("tdSigT"))
```

(continues on next page)

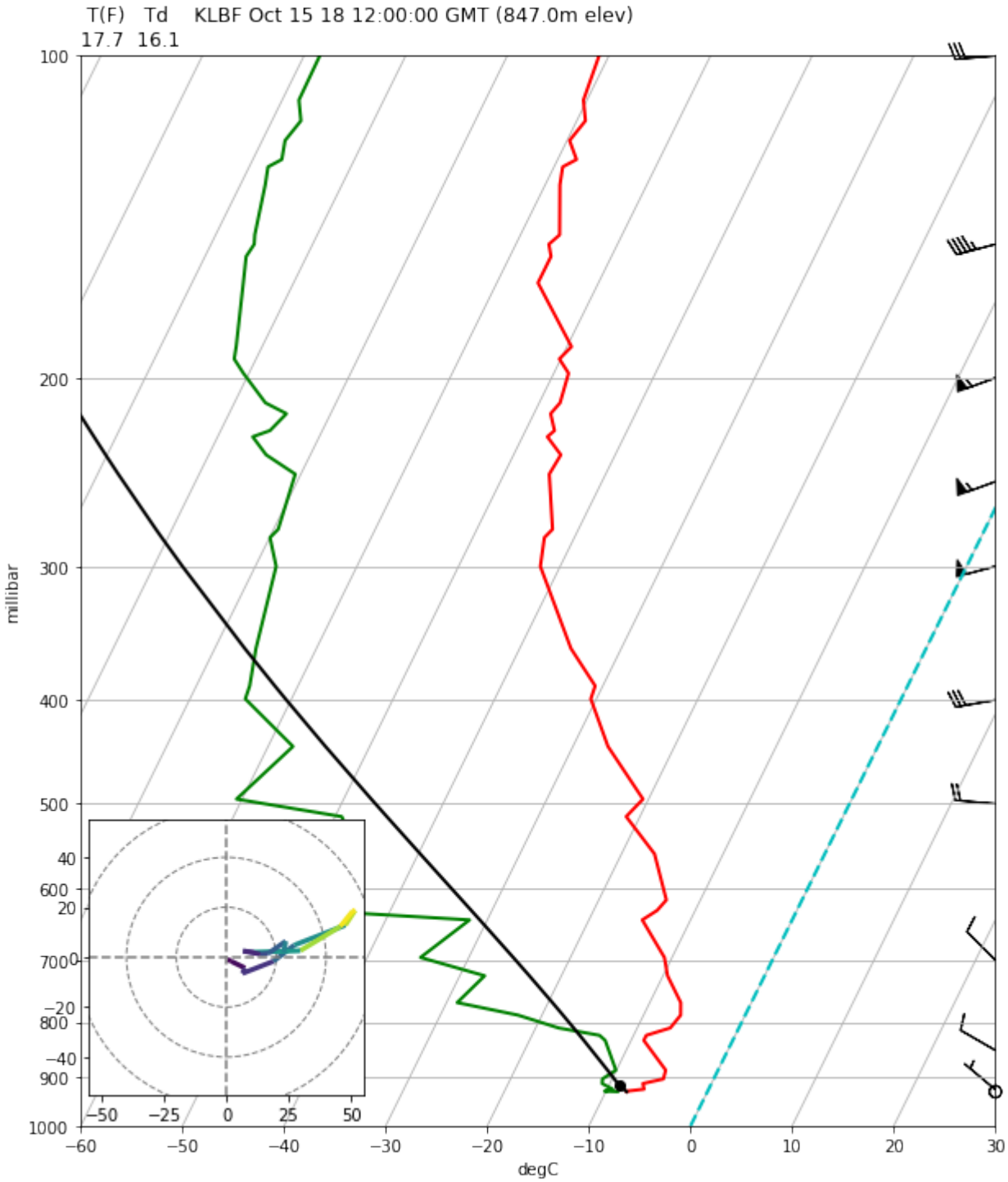
(continued from previous page)

```
        continue

# Sort mandatory levels (but not sigT levels) because of the 1000.MB interpolation␣
↪inclusion
ps = prMan.argsort()[::-1]
wpres = prMan[ps]
direc = wdMan[ps]
spd   = wsMan[ps]
tman = tpMan[ps]
dman = tdMan[ps]

# Flag missing data
prSig[prSig <= -9999] = np.nan
tpSig[tpSig <= -9999] = np.nan
tdSig[tdSig <= -9999] = np.nan
wpres[wpres <= -9999] = np.nan
tman[tman <= -9999] = np.nan
dman[dman <= -9999] = np.nan
direc[direc <= -9999] = np.nan
spd[spd <= -9999] = np.nan

# assign units
p = (prSig/100) * units.mbar
wpres = (wpres/100) * units.mbar
u,v = wind_components(spd, np.deg2rad(direc))

if tpUnit is 'K':
    T = (tpSig-273.15) * units.degC
    Td = (tdSig-273.15) * units.degC
    tman = tman * units.degC
    dman = dman * units.degC

# Create SkewT/LogP
plt.rcParams['figure.figsize'] = (10, 12)
skew = SkewT()
skew.plot(p, T, 'r', linewidth=2)
skew.plot(p, Td, 'g', linewidth=2)
skew.plot_barbs(wpres, u, v)
skew.ax.set_ylim(1000, 100)
skew.ax.set_xlim(-60, 30)

title_string = " T(F)   Td   "
title_string +=  " " + str(ob.getString("staName"))
title_string += " " + str(ob.getDataTime().getRefTime())
title_string += " (" + str(ob.getNumber("staElev")) + "m elev)"
title_string += "\n" + str(round(T[0].to('degF').item(),1))
title_string += "  " + str(round(Td[0].to('degF').item(),1))
plt.title(title_string, loc='left')

# Calculate LCL height and plot as black dot
lcl_pressure, lcl_temperature = lcl(p[0], T[0], Td[0])
skew.plot(lcl_pressure, lcl_temperature, 'ko', markerfacecolor='black')

# Calculate full parcel profile and add to plot as black line
prof = parcel_profile(p, T[0], Td[0]).to('degC')
skew.plot(p, prof, 'k', linewidth=2)
```

(continues on next page)

(continued from previous page)

```
# An example of a slanted line at constant T -- in this case the 0 isotherm
l = skew.ax.axvline(0, color='c', linestyle='--', linewidth=2)

# Draw hodograph
ax_hod = inset_axes(skew.ax, '30%', '30%', loc=3)
h = Hodograph(ax_hod, component_range=max(wsMan))
h.add_grid(increment=20)
h.plot_colormapped(u, v, spd)

# Show the plot
plt.show()
```



Watch and Warning Polygons

Notebook

This example uses matplotlib, cartopy, shapely, and python-awips to plot watch and warning polygons requested from

a real-time AWIPS EDEX server.

First, set up our imports and define functions to be used later:

```
from awips.dataaccess import DataAccessLayer
from awips.tables import vtec
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
from cartopy.feature import ShapelyFeature,NaturalEarthFeature
from shapely.geometry import MultiPolygon,Polygon

def warning_color(phensig):
    return vtec[phensig]['color']

def make_map(bbox, projection=ccrs.PlateCarree()):
    fig, ax = plt.subplots(figsize=(20,12),
            subplot_kw=dict(projection=projection))
    ax.set_extent(bbox)
    gl = ax.gridlines(draw_labels=True)
    gl.xlabels_top = gl.ylabels_right = False
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    return fig, ax
```

Next, we create a request for the "warning" data type:

```
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
request = DataAccessLayer.newDataRequest()
request.setDatatype("warning")
request.setParameters('phensig')
times = DataAccessLayer.getAvailableTimes(request)

# Get records for last 50 available times
response = DataAccessLayer.getGeometryData(request, times[-50:-1])
print("Using " + str(len(response)) + " records")

# Each record will have a numpy array the length of the number of "parameters"
# Default is 1 (request.setParameters('phensig'))
parameters = {}
for x in request.getParameters():
    parameters[x] = np.array([])
print(parameters)
```

```
Using 109 records
{'phensig': array([], dtype=float64)}
```

Now loop through each record and plot it as either Polygon or MultiPolygon, with appropriate colors

```
%matplotlib inline
bbox=[-127,-64,24,49]
fig, ax = make_map(bbox=bbox)

siteids=np.array([])
periods=np.array([])
```

(continues on next page)

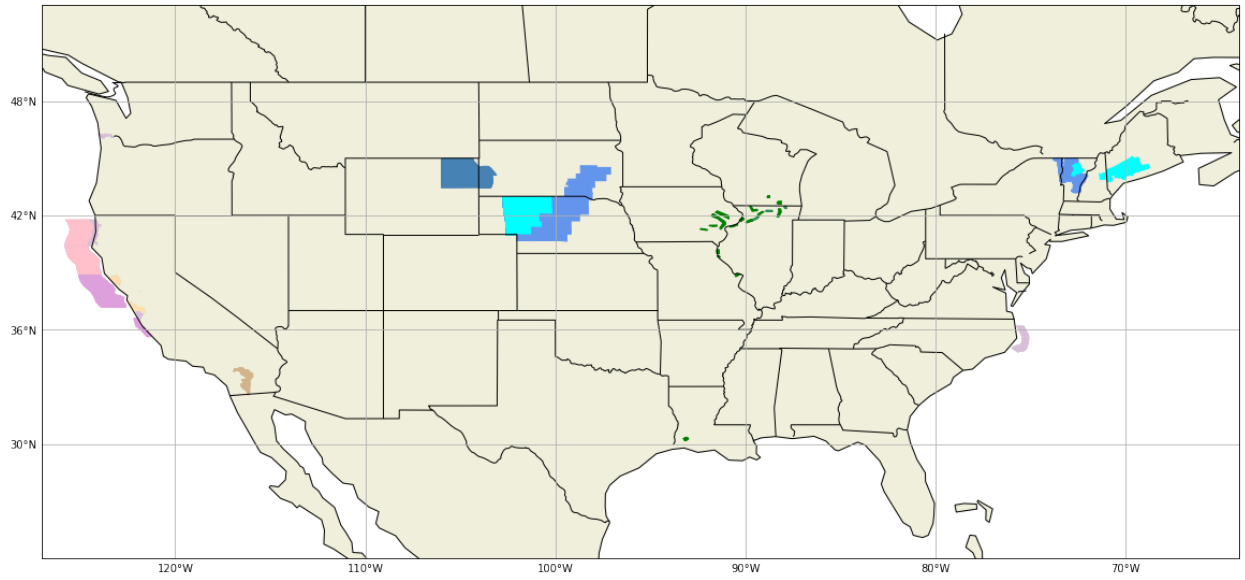
(continued from previous page)

```
reftimes=np.array([])

for ob in response:

    poly = ob.getGeometry()
    site = ob.getLocationName()
    pd   = ob.getDataTime().getValidPeriod()
    ref  = ob.getDataTime().getRefTime()

    # do not plot if phensig is blank (SPS)
    if ob.getString('phensig'):

        phensigString = ob.getString('phensig')

        siteids = np.append(siteids,site)
        periods = np.append(periods,pd)
        reftimes = np.append(reftimes,ref)

        for parm in parameters:
            parameters[parm] = np.append(parameters[parm],ob.getString(parm))

        if poly.geom_type == 'MultiPolygon':
            geometries = np.array([])
            geometries = np.append(geometries,MultiPolygon(poly))
            geom_count = ", " + str(len(geometries)) +" geometries"
        else:
            geometries = np.array([])
            geometries = np.append(geometries,Polygon(poly))
            geom_count=""

        for geom in geometries:
            bounds = Polygon(geom)
            intersection = bounds.intersection
            geoms = (intersection(geom)
                 for geom in geometries
                 if bounds.intersects(geom))

        #print(vtec[phensigString]['hdln']
        #  + " (" + phensigString + ") issued at " + str(ref)
        #  + " ("+str(poly.geom_type) + geom_count + ")")

        color = warning_color(phensigString)
        shape_feature = ShapelyFeature(geoms,ccrs.PlateCarree(),
                        facecolor=color, edgecolor=color)
        ax.add_feature(shape_feature)

states_provinces = cfeature.NaturalEarthFeature(
        category='cultural',
        name='admin_1_states_provinces_lines',
        scale='50m',
        facecolor='none')
political_boundaries = cfeature.NaturalEarthFeature(category='cultural',
                               name='admin_0_boundary_lines_land',
                               scale='50m', facecolor='none')
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(states_provinces, edgecolor='black')
```

(continues on next page)

(continued from previous page)

```
ax.add_feature(political_boundaries, edgecolor='black')

plt.show()
```



Development Guide

The Data Access Framework allows developers to retrieve different types of data without having dependencies on those types of data. It provides a single, unified data type that can be customized by individual implementing plug-ins to provide full functionality pertinent to each data type.

Writing a New Factory

Factories will most often be written in a dataplugin, but should always be written in a common plug-in. This will allow for clean dependencies from both CAVE and EDEX.

A new plug-in's data access class must implement IDataFactory. For ease of use, abstract classes have been created to combine similar methods. Data factories do not have to implement both types of data (grid and geometry). They can if they choose, but if they choose not to, they should do the following:

```
throw new UnsupportedOutputTypeException(request.getDatatype(), "grid");
```

This lets the code know that grid type is not supported for this data factory. Depending on where the data is coming from, helpers have been written to make writing a new data type factory easier. For example, PluginDataObjects can use AbstractDataPluginFactory as a start and not have to create everything from scratch.

Each data type is allowed to implement retrieval in any manner that is felt necessary. The power of the framework means that the code retrieving data does not have to know anything of the underlying retrieval methods, only that it is getting data in a certain manner. To see some examples of ways to retrieve data, reference **SatelliteGridFactory** and **RadarGridFactory**.

Methods required for implementation:

public DataTime[] getAvailableTimes(IDataRequest request)

- This method returns an array of DataTime objects corresponding to what times are available for the data being retrieved, based on the parameters and identifiers being passed in.

public DataTime[] getAvailableTimes(IDataRequest request, BinOffset binOffset)

- This method returns available times as above, only with a bin offset applied.

Note: Both of the preceding methods can throw TimeAgnosticDataException exceptions if times do not apply to the data type.

public IGridData[] getGridData(IDataRequest request, DataTime...times)

- This method returns IGridData objects (an array) based on the request and times to request for. There can be multiple times or a single time.

public IGridData[] getGridData(IDataRequest request, TimeRange range)

- Similar to the preceding method, this returns IGridData objects based on a range of times.

public IGeometryData[] getGeometryData(IDataRequest request, DataTime times)

- This method returns IGeometryData objects based on a request and times.

public IGeometryData[] getGeometryData(IDataRequest request, TimeRange range)

- Like the preceding method, this method returns IGeometryData objects based on a range of times.

public String[] getAvailableLocationNames(IDataRequest request)

- This method returns location names that match the request. If this does not apply to the data type, an IncompatibleRequestException should be thrown.

Registering the Factory with the Framework

The following needs to be added in a spring file in the plug-in that contains the new factory:

```
<bean id="radarGridFactory"
    class="com.raytheon.uf.common.dataplugin.radar.dataaccess.RadarGridFactory" />
<bean factory-bean="dataAccessRegistry" factorymethod="register">
    <constructor-arg value="radar"/>
    <constructor-arg ref="radarGridFactory"/>
</bean>
```

This takes the RadarGridFactory and registers it with the registry and allows it to be used any time the code makes a request for the data type "radar."

Retrieving Data Using the Factory

For ease of use and more diverse use, there are multiple interfaces into the Data Access Layer. Currently, there is a Python implementation and a Java implementation, which have very similar method calls and work in a similar manner. Plug-ins that want to use the data access framework to retrieve data should include **com.raytheon.uf.common.dataaccess** as a Required Bundle in their MANIFEST.MF.

To retrieve data using the Python interface :

```
from awips.dataaccess import DataAccessLayer
req = DataAccessLayer.newDataRequest()
req.setDatatype("grid")
req.setParameters("T")
```

(continues on next page)

(continued from previous page)

```
req.setLevels("2FHAG")
req.addIdentifier("info.datasetId", "GFS40")
times = DataAccessLayer.getAvailableTimes(req)
data = DataAccessLayer.getGridData(req, times)
```

To retrieve data using the Java interface :

```
IDataRequest req = DataAccessLayer.newDataRequest();
req.setDatatype("grid");
req.setParameters("T");
req.setLevels("2FHAG");
req.addIdentifier("info.datasetId", "GFS40");
DataTime[] times = DataAccessLayer.getAvailableTimes(req)
IData data = DataAccessLayer.getGridData(req, times);
```

newDataRequest()

- This creates a new data request. Most often this is a DefaultDataRequest, but saves for future implentations as well.

setDatatype(String)

- This is the data type being retrieved. This can be found as the value that is registered when creating the new factory (See section above **Registering the Factory with the Framework** [radar in that case]).

setParameters(String...)

- This can differ depending on data type. It is most often used as a main difference between products.

setLevels(String...)

- This is often used to identify the same products on different mathematical angles, heights, levels, etc.

addIdentifier(String, String)

- This differs based on data type, but is often used for more fine-tuned querying.

Both methods return a similar set of data and can be manipulated by their respective languages. See DataAccessLayer.py and DataAccessLayer.java for more methods that can be called to retrieve data and different parts of the data. Because each data type has different parameters, levels, and identifiers, it is best to see the actual data type for the available options. If it is undocumented, then the best way to identify what parameters are to be used is to reference the code.

Development Background

In support of Hazard Services Raytheon Technical Services is building a generic data access framework that can be called via JAVA or Python. The data access framework code can be found within the AWIPS Baseline in

```
com.raytheon.uf.common.dataaccess
```

As of 2016, plugins have been written for grid, radar, satellite, Hydro (SHEF), point data (METAR, SYNOP, Profiler, ACARS, AIREP, PIREP), maps data, and other data types. The Factories for each can be found in the following packages (you may need to look at the development baseline to see these):

```
com.raytheon.uf.common.dataplugin.grid.dataaccess
com.raytheon.uf.common.dataplugin.radar.dataaccess
com.raytheon.uf.common.dataplugin.satellite.dataaccess
com.raytheon.uf.common.dataplugin.binlightning.dataaccess
```

(continues on next page)

(continued from previous page)

```
com.raytheon.uf.common.dataplugin.sfc.dataaccess
com.raytheon.uf.common.dataplugin.sfcobs.dataaccess
com.raytheon.uf.common.dataplugin.acars.dataaccess
com.raytheon.uf.common.dataplugin.ffmp.dataaccess
com.raytheon.uf.common.dataplugin.bufrua.dataaccess
com.raytheon.uf.common.dataplugin.profiler.dataaccess
com.raytheon.uf.common.dataplugin.moddelsounding.dataaccess
com.raytheon.uf.common.dataplugin.ldadmesonet.dataaccess
com.raytheon.uf.common.dataplugin.binlightning.dataaccess
com.raytheon.uf.common.dataplugin.gfe.dataaccess
com.raytheon.uf.common.hydro.dataaccess
com.raytheon.uf.common.pointdata.dataaccess
com.raytheon.uf.common.dataplugin.maps.dataaccess
```

Additional data types may be added in the future. To determine what datatypes are supported display the "type hierarchy" associated with the classes

AbstractGridDataPluginFactory,

AbstractGeometryDatabaseFactory, and

AbstractGeometryTimeAgnosticDatabaseFactory.

The following content was taken from the design review document which is attached and modified slightly.

Design/Implementation

The Data Access Framework is designed to provide a consistent interface for requesting and using geospatial data within CAVE or EDEX. Examples of geospatial data are grids, satellite, radar, metars, maps, river gage heights, FFMP basin data, airmets, etc. To allow for convenient use of geospatial data, the framework will support two types of requests: grids and geometries (points, polygons, etc). The framework will also hide implementation details of specific data types from users, making it easier to use data without worrying about how the data objects are structured or retrieved.

A suggested mapping of some current data types to one of the two supported data requests is listed below. This list is not definitive and can be expanded. If a developer can dream up an interpretation of the data in the other supported request type, that support can be added.

Grids

- Grib
- Satellite
- Radar
- GFE

Geometries

- Map (states, counties, zones, etc)
- Hydro DB (IHFS)
- Obs (metar)
- FFMP
- Hazard
- Warning

- CCFP
- Airmet

The framework is designed around the concept of each data type plugin contributing the necessary code for the framework to support its data. For example, the satellite plugin provides a factory class for interacting with the framework and registers itself as being compatible with the Data Access Framework. This concept is similar to how EDEX in AWIPS expects a plugin developer to provide a decoder class and record class and register them, but then automatically manages the rest of the ingest process including routing, storing, and alerting on new data. This style of plugin architecture effectively enables the framework to expand its capabilities to more data types without having to alter the framework code itself. This will enable software developers to incrementally add support for more data types as time allows, and allow the framework to expand to new data types as they become available.

The Data Access Framework will not break any existing functionality or APIs, and there are no plans to retrofit existing cosde to use the new API at this time. Ideally code will be retrofitted in the future to improve ease of maintainability. The plugin pecific code that hooks into the framework will make use of existing APIs such as **IDataStore** and **IServerRequest** to complete the requests.

The Data Access Framework can be understood as three parts:

- How users of the framework retrieve and use the data
- How plugin developers contribute support for new data types
- How the framework works when it receives a request

How users of the framework retrieve and use the data

When a user of the framework wishes to request data, they must instantiate a request object and set some of the values on that request. Two request interfaces will be supported, for detailed methods see section "Detailed Code" below.

IDataRequest

IGridRequest extends **IDataRequest**

IGeometryRequest extends **IDataRequest**

For the request interfaces, default implementations of **DefaultGridRequest** and **DefaultGeometryRequest** will be provided to handle most cases. However, the use of interfaces allows for custom special cases in the future. If necessary, the developer of a plugin can write their own custom request implementation to handle a special case.

After the request object has been prepared, the user will pass it to the Data Access Layer to receive a data object in return. See the "Detailed Code" section below for detailed methods of the Data Access Layer. The Data Access Layer will return one of two data interfaces.

IData

IGridData extends **IData**

IGeometryData extends **IData**

For the data interfaces, the use of interfaces effectively hides the implementation details of specific data types from the user of the framework. For example, the user receives an **IGridData** and knows the data time, grid geometry, parameter, and level, but does not know that the data is actually a **GFEGridData** vs **D2DGridData** vs **SatelliteGridData**. This enables users of the framework to write generic code that can support multiple data types.

For python users of the framework, the interfaces will be very similar with a few key distinctions. Geometries will be represented by python geometries from the open source Shapely project. For grids, the python **IGridData** will have a method for requesting the raw data as a numpy array, and the Data Access Layer will have methods for requesting the latitude coordinates and the longitude coordinates of grids as numpy arrays. The python requests and data objects will be pure python and not JEP PyJObjects that wrap Java objects. A future goal of the Data Access Framework is to

provide support to python local apps and therefore enable requests of data outside of CAVE and EDEX to go through the same familiar interfaces. This goal is out of scope for this project but by making the request and returned data objects pure python it will not be a huge undertaking to add this support in the future.

How plugin developers contribute support for new datatypes

When a developer wishes to add support for another data type to the framework, they must implement one or both of the factory interfaces within a common plugin. Two factory interfaces will be supported, for detailed methods see below.

IDataFactory

IGridFactory extends **IDataFactory**

IGeometryFactory extends **IDataFactory**

For some data types, it may be desired to add support for both types of requests. For example, the developer of grid data may want to provide support for both grid requests and geometry requests. In this case the developer would write two separate classes where one implements **IGridFactory** and the other implements **IGeometryFactory**. Furthermore, factories could be stacked on top of one another by having factory implementations call into the Data Access Layer.

For example, a custom factory keyed to "derived" could be written for derived parameters, and the implementation of that factory may then call into the Data Access Layer to retrieve "grid" data. In this example the raw data would be retrieved through the **GridDataFactory** while the derived factory then applies the calculations before returning the data.

Implementations do not need to support all methods on the interfaces or all values on the request objects. For example, a developer writing the **MapGeometryFactory** does not need to support **getAvailableTimes()** because map data such as US counties is time agnostic. In this case the method should throw **UnsupportedOperationException** and the javadoc will indicate this.

Another example would be the developer writing **ObsGeometryFactory** can ignore the Level field of the **IDataRequest** as there are not different levels of metar data, it is all at the surface. It is up to the factory writer to determine which methods and fields to support and which to ignore, but the factory writer should always code the factory with the user requesting data in mind. If a user of the framework could reasonably expect certain behavior from the framework based on the request, the factory writer should implement support for that behavior.

Abstract factories will be provided and can be extended to reduce the amount of code a factory developer has to write to complete some common actions that will be used by multiple factories. The factory should be capable of working within either CAVE or EDEX, therefore all of its server specific actions (e.g. database queries) should go through the Request/Handler API by using **IServerRequests**. CAVE can then send the **IServerRequests** to EDEX with **ThriftClient** while EDEX can use the **ServerRequestRouter** to process the **IServerRequests**, making the code compatible regardless of which JVM it is running inside.

Once the factory code is written, it must be registered with the framework as an available factory. This will be done through spring xml in a common plugin, with the xml file inside the res/spring folder of the plugin. Registering the factory will identify the datatype name that must match what users would use as the datatype on the **IDataRequest**, e.g. the word "satellite". Registering the factory also indicates to the framework what request types are supported, i.e. grid vs geometry or both.

An example of the spring xml for a satellite factory is provided below:

```
<bean id="satelliteFactory"
  class="com.raytheon.uf.common.dataplugin.satellite.SatelliteFactory" />

<bean id="satelliteFactoryRegistered" factory-bean="dataFactoryRegistry" factory-
↪method="register">
    <constructor-arg value="satellite" />
```

(continues on next page)

(continued from previous page)

```
    <constructor-arg value="com.raytheon.uf.common.dataaccess.grid.IGridRequest" />
    <constructor-arg value="satelliteFactory" />
</bean>
```

How the framework works when it receives a request

IDataRequest requires a datatype to be set on every request. The framework will have a registry of existing factories for each data type (grid and geometry). When the Data Access Layer methods are called, it will first lookup in the registry for the factory that corresponds to the datatype on the **IDataRequest**. If no corresponding factory is found, it will throw an exception with a useful error message that indicates there is no current support for that datatype request. If a factory is found, it will delegate the processing of the request to the factory. The factory will receive the request and process it, returning the result back to the Data Access Layer which then returns it to the caller.

By going through the Data Access Layer, the user is able to retrieve the data and use it without understanding which factory was used, how the factory retrieved the data, or what implementation of data was returned. This effectively frees the framework and users of the framework from any dependencies on any particular data types. Since these dependencies are avoided, the specific **IDataFactory** and **IData** implementations can be altered in the future if necessary and the code making use of the framework will not need to be changed as long as the interfaces continue to be met.

Essentially, the Data Access Framework is a service that provides data in a consistent way, with the service capabilities being expanded by plugin developers who write support for more data types. Note that the framework itself is useless without plugins contributing and registering **IDataFactories**. Once the framework is coded, developers will need to be tasked to add the factories necessary to support the needed data types.

Request interfaces

Requests and returned data interfaces will exist in both Java and Python. The Java interfaces are listed below and the Python interfaces will match the Java interfaces except where noted. Factories will only be written in Java.

IDataRequest

- **void setDatatype(String datatype)** - the datatype name and also the key to which factory will be used. Frequently pluginName such as radar, satellite, gfe, ffmp, etc
- **void addIdentifier(String key, Object value)** - an identifier the factory can use to determine which data to return, e.g. for grib data key "modelName" and value "GFS40"
- **void setParameters(String... params)**
- **void setLevels(Level... levels)**
- **String getDatatype()**
- **Map getIdentifiers()**
- **String[] getParameters()**
- **Level[] getLevels()**
- Python Differences
- **Levels** will be represented as **Strings**

IGridRequest extends IDataRequest

- **void setStorageRequest(Request request)** - a datastorage request that allows for slab, line, and point requests for faster performance and less data retrieval

- **Request getStorageRequest()**
- Python Differences
- No support for storage requests

IGeometryRequest extends IDataRequest

- **void setEnvelope(Envelope env)** - a bounding box envelope to limit the data that is searched through and returned. Not all factories may support this.
- **setLocationNames(String... locationNames)** - a convenience of requesting data by names such as ICAOs, airports, stationIDs, etc
- **Envelope getEnvelope()**
- **String[] getLocationNames()**
- Python Differences
- Envelope methods will use a **shapely.geometry.Polygon** instead of **Envelopes** (shapely has no concept of envelopes and considers them as rectangular polygons)

Data Interfaces

IData

- **Object getAttribute(String key)** - **getAttribute** provides a way to get at attributes of the data that the interface does not provide, allowing the user to get more info about the data without adding dependencies on the specific data type plugin
- **DataTime getDataTime()** - some data may return null (e.g. maps)
- **Level getLevel()** - some data may return null
- Python Differences
- **Levels** will be represented by **Strings**

IGridData extends IData

- **String getParameter()**
- **GridGeometry2D getGridGeometry()**
- **Unit getUnit()** - some data may return null
- **DataDestination populateData(DataDestination destination)** - How the user gets the raw data by passing in a **DataDestination** such as **FloatArrayWrapper** or **ByteBufferWrapper**. This allows the user to specify the way the raw data of the grid should be structured in memory.
- **DataDestination populateData(DataDestination destination, Unit unit)** - Same as the above method but also attempts to convert the raw data to the specified unit when populating the **DataDestination**.
- Python Differences
- **Units** will be represented by **Strings**
- **populateData()** methods will not exist, instead there will be a **getRawData()** method that returns a numpy array in the native type of the data

IGeometryData extends IData

- **Geometry getGeometry()**
- **Set getParameters()** - Gets the list of parameters included in this data

- **String getString(String param)** - Gets the value of the parameter as a String
- **Number getNumber(String param)** - Gets the value of the parameter as a Number
- **Unit getUnit(String param)** - Gets the unit of the parameter, may be null
- **Type getType(String param)** - Returns an enum of the raw type of the parameter, such as Float, Int, or String
- **String getLocationName()** - Returns the location name of the piece of data, typically to correlate if the request was made with locationNames. May be null.
- Python Differences
- **Geometry** will be **shapely.geometry.Geometry**
- **getNumber()** will return the python native number of the data
- **Units** will be represented by **Strings**
- **getType()** will return the python type object

DataAccessLayer (in implementation, these methods delegate processing to factories)

- **DataTime[] getAvailableTimes(IDataRequest request)**
- **DataTime[] getAvailableTimes(IDataRequest request, BinOffset binOffset)**
- **IData[] getData(IDataRequest request, DataTime... times)**
- **IData[] getData(IDataRequest request, TimeRange timeRange)**
- **GridGeometry2D getGridGeometry(IGridRequest request)**
- **String[] getAvailableLocationNames(IGeometryRequest request)**
- Python Differences
- No support for **BinOffset**
- **getGridGeometry(IGridRequest)** will be replaced by **getLatCoords(IGridRequest)** and **getLonCoords(IGridRequest)** that will return numpy arrays of the lat or lon of every grid cell

Factory Interfaces (Java only)

- **IDataFactory**
- **DataTime[] getAvailableTimes(R request)** - queries the database and returns the times that match the request. Some factories may not support this (e.g. maps).
- **DataTime[] getAvailableTimes(R request, BinOffset binOffset)** - queries the database with a bin offset and returns the times that match the request. Some factories may not support this.
- **D[] getData(R request, DataTime... times)** - Gets the data that matches the request at the specified times.
- **D[] getData(R request, TimeRange timeRange)** - Gets the data that matches the request and is within the time range.

IGridDataFactory extends IDataFactory

- **GridGeometry2D getGeometry(IGridRequest request)** - Returns the grid geometry of the data that matches the request BEFORE making the request. Useful for then making slab or line requests for subsets of the data. Does not support moving grids, but moving grids don't make subset requests either.

IGeometryDataFactory extends IDataFactory

- **getAvailableLocationNames(IGeometryRequest request)** - Convenience method to retrieve available location names that match a request. Not all factories may support this.

Grid Parameters

Abbreviation	Description
WSPD	10 Metre neutral wind speed over waves
WDRT	10 Metre Wind Direction Over Waves
ARI12H1000YR	12H Average Recurrance Interval Accumulation 1000 Year
ARI12H100YR	12H Average Recurrance Interval Accumulation 100 Year
ARI12H10YR	12H Average Recurrance Interval Accumulation 10 Year
ARI12H1YR	12H Average Recurrance Interval Accumulation 1 Year
ARI12H200YR	12H Average Recurrance Interval Accumulation 200 Year
ARI12H25YR	12H Average Recurrance Interval Accumulation 25 Year
ARI12H2YR	12H Average Recurrance Interval Accumulation 2 Year
ARI12H500YR	12H Average Recurrance Interval Accumulation 500 Year
ARI12H50YR	12H Average Recurrance Interval Accumulation 50 Year
ARI12H5YR	12H Average Recurrance Interval Accumulation 5 Year
PRP12H	12 hour Precipitation Accumulation Return Period
GaugeInfIndex12H	12 hour QPE Gauge Influence Index
FFG12	12-hr flash flood guidance
FFR12	12-hr flash flood runoff values
EchoTop18	18 dBZ Echo Top
ARI1H1000YR	1H Average Recurrance Interval Accumulation 1000 Year
ARI1H100YR	1H Average Recurrance Interval Accumulation 100 Year
ARI1H10YR	1H Average Recurrance Interval Accumulation 10 Year
ARI1H1YR	1H Average Recurrance Interval Accumulation 1 Year
ARI1H200YR	1H Average Recurrance Interval Accumulation 200 Year
ARI1H25YR	1H Average Recurrance Interval Accumulation 25 Year
ARI1H2YR	1H Average Recurrance Interval Accumulation 2 Year
ARI1H500YR	1H Average Recurrance Interval Accumulation 500 Year
ARI1H50YR	1H Average Recurrance Interval Accumulation 50 Year
ARI1H5YR	1H Average Recurrance Interval Accumulation 5 Year
PRP01H	1 hour Precipitation Accumulation Return Period
GaugeInfIndex01H	1 hour QPE Gauge Influence Index
QPEFFG01H	1 hour QPE-to-FFG Ratio
FFG01	1-hr flash flood guidance
FFR01	1-hr flash flood runoff values
QPE01	1-hr Quantitative Precip Estimate
QPE01_ACR	1-hr Quantitative Precip Estimate
QPE01_ALR	1-hr Quantitative Precip Estimate
QPE01_FWR	1-hr Quantitative Precip Estimate
QPE01_KRF	1-hr Quantitative Precip Estimate
QPE01_MSR	1-hr Quantitative Precip Estimate
QPE01_ORN	1-hr Quantitative Precip Estimate
QPE01_PTR	1-hr Quantitative Precip Estimate
QPE01_RHA	1-hr Quantitative Precip Estimate
QPE01_RSA	1-hr Quantitative Precip Estimate
QPE01_STR	1-hr Quantitative Precip Estimate
QPE01_TAR	1-hr Quantitative Precip Estimate

Table 1 – continued from previo

Abbreviation	Description
QPE01_TIR	1-hr Quantitative Precip Estimate
QPE01_TUA	1-hr Quantitative Precip Estimate
EVEC1	1st Vector Component of Electric Field
BVEC1	1st Vector Component of Magnetic Field
VEL1	1st Vector Component of Velocity (Coordinate system dependent)
TCSRG20	20% Tropical Cyclone Storm Surge Exceedance
ARI24H1000YR	24H Average Recurrance Interval Accumulation 1000 Year
ARI24H100YR	24H Average Recurrance Interval Accumulation 100 Year
ARI24H10YR	24H Average Recurrance Interval Accumulation 10 Year
ARI24H1YR	24H Average Recurrance Interval Accumulation 1 Year
ARI24H200YR	24H Average Recurrance Interval Accumulation 200 Year
ARI24H25YR	24H Average Recurrance Interval Accumulation 25 Year
ARI24H2YR	24H Average Recurrance Interval Accumulation 2 Year
ARI24H500YR	24H Average Recurrance Interval Accumulation 500 Year
ARI24H50YR	24H Average Recurrance Interval Accumulation 50 Year
ARI24H5YR	24H Average Recurrance Interval Accumulation 5 Year
PRP24H	24 hour Precipitation Accumulation Return Period
GaugeInfIndex24H	24 hour QPE Gauge Influence Index
FFG24	24-hr flash flood guidance
FFR24	24-hr flash flood runoff values
QPE24	24-hr Quantitative Precip Estimate
QPE24_ACR	24-hr Quantitative Precip Estimate
QPE24_ALR	24-hr Quantitative Precip Estimate
QPE24_FWR	24-hr Quantitative Precip Estimate
QPE24_KRF	24-hr Quantitative Precip Estimate
QPE24_MSR	24-hr Quantitative Precip Estimate
QPE24_ORN	24-hr Quantitative Precip Estimate
QPE24_PTR	24-hr Quantitative Precip Estimate
QPE24_RHA	24-hr Quantitative Precip Estimate
QPE24_RSA	24-hr Quantitative Precip Estimate
QPE24_STR	24-hr Quantitative Precip Estimate
QPE24_TAR	24-hr Quantitative Precip Estimate
QPE24_TIR	24-hr Quantitative Precip Estimate
QPE24_TUA	24-hr Quantitative Precip Estimate
QPF24	24-hr Quantitative Precip Forecast
QPF24_ACR	24-hr Quantitative Precip Forecast
QPF24_ALR 205	24-hr Quantitative Precip Forecast
QPF24_FWR	24-hr Quantitative Precip Forecast
QPF24_KRF	24-hr Quantitative Precip Forecast
QPF24_MSR	24-hr Quantitative Precip Forecast
QPF24_ORN	24-hr Quantitative Precip Forecast
QPF24_PTR	24-hr Quantitative Precip Forecast
QPF24_RHA	24-hr Quantitative Precip Forecast
QPF24_RSA	24-hr Quantitative Precip Forecast
QPF24_STR	24-hr Quantitative Precip Forecast
QPF24_TAR	24-hr Quantitative Precip Forecast
QPF24_TIR	24-hr Quantitative Precip Forecast
QPF24_TUA	24-hr Quantitative Precip Forecast
ARI2H1000YR	2H Average Recurrance Interval Accumulation 1000 Year

Table 1 – continued from previo

Abbreviation	Description
ARI2H100YR	2H Average Recurrance Interval Accumulation 100 Year
ARI2H10YR	2H Average Recurrance Interval Accumulation 10 Year
ARI2H1YR	2H Average Recurrance Interval Accumulation 1 Year
ARI2H200YR	2H Average Recurrance Interval Accumulation 200 Year
ARI2H25YR	2H Average Recurrance Interval Accumulation 25 Year
ARI2H2YR	2H Average Recurrance Interval Accumulation 2 Year
ARI2H500YR	2H Average Recurrance Interval Accumulation 500 Year
ARI2H50YR	2H Average Recurrance Interval Accumulation 50 Year
ARI2H5YR	2H Average Recurrance Interval Accumulation 5 Year
EVEC2	2nd Vector Component of Electric Field
BVEC2	2nd Vector Component of Magnetic Field
VEL2	2nd Vector Component of Velocity (Coordinate system dependent)
EchoTop30	30 dBZ Echo Top
ARI30M1000YR	30M Average Recurrance Interval Accumulation 1000 Year
ARI30M100YR	30M Average Recurrance Interval Accumulation 100 Year
ARI30M10YR	30M Average Recurrance Interval Accumulation 10 Year
ARI30M1YR	30M Average Recurrance Interval Accumulation 1 Year
ARI30M200YR	30M Average Recurrance Interval Accumulation 200 Year
ARI30M25YR	30M Average Recurrance Interval Accumulation 25 Year
ARI30M2YR	30M Average Recurrance Interval Accumulation 2 Year
ARI30M500YR	30M Average Recurrance Interval Accumulation 500 Year
ARI30M50YR	30M Average Recurrance Interval Accumulation 50 Year
ARI30M5YR	30M Average Recurrance Interval Accumulation 5 Year
PRP30min	30 min Precipitation Accumulation Return Period
TCSRG30	30% Tropical Cyclone Storm Surge Exceedance
SALIN	3-D Salinity
WTMPC	3-D Temperature
ARI3H1000YR	3H Average Recurrance Interval Accumulation 1000 Year
ARI3H100YR	3H Average Recurrance Interval Accumulation 100 Year
ARI3H10YR	3H Average Recurrance Interval Accumulation 10 Year
ARI3H1YR	3H Average Recurrance Interval Accumulation 1 Year
ARI3H200YR	3H Average Recurrance Interval Accumulation 200 Year
ARI3H25YR	3H Average Recurrance Interval Accumulation 25 Year
ARI3H2YR	3H Average Recurrance Interval Accumulation 2 Year
ARI3H500YR	3H Average Recurrance Interval Accumulation 500 Year
ARI3H50YR	3H Average Recurrance Interval Accumulation 50 Year
ARI3H5YR	3H Average Recurrance Interval Accumulation 5 Year
PRP03H	3 hour Precipitation Accumulation Return Period
GaugeInfIndex03H	3 hour QPE Gauge Influence Index
QPEFFG03H	3 hour QPE-to-FFG Ratio
FFG03	3-hr flash flood guidance
FFR03	3-hr flash flood runoff values
TSLSA	3-hr pressure tendency (Std. Atmos. Reduction)
EVEC3	3rd Vector Component of Electric Field
BVEC3	3rd Vector Component of Magnetic Field
VEL3	3rd Vector Component of Velocity (Coordinate system dependent)
TCSRG40	40% Tropical Cyclone Storm Surge Exceedance
GaugeInfIndex48H	48 hour QPE Gauge Influence Index
EchoTop50	50 dBZ Echo Top

Table 1 – continued from previo

Abbreviation	Description
TCSRG50	50% Tropical Cyclone Storm Surge Exceedance
5WAVA	5-wave geopotential height anomaly
5WAVA	5-Wave Geopotential Height Anomaly
5WAVH	5-wave geopotential height
5WAVH	5-Wave Geopotential Height
EchoTop60	60 dBZ Echo Top
TCSRG60	60% Tropical Cyclone Storm Surge Exceedance
ARI6H1000YR	6H Average Recurrance Interval Accumulation 1000 Year
ARI6H100YR	6H Average Recurrance Interval Accumulation 100 Year
ARI6H10YR	6H Average Recurrance Interval Accumulation 10 Year
ARI6H1YR	6H Average Recurrance Interval Accumulation 1 Year
ARI6H200YR	6H Average Recurrance Interval Accumulation 200 Year
ARI6H25YR	6H Average Recurrance Interval Accumulation 25 Year
ARI6H2YR	6H Average Recurrance Interval Accumulation 2 Year
ARI6H500YR	6H Average Recurrance Interval Accumulation 500 Year
ARI6H50YR	6H Average Recurrance Interval Accumulation 50 Year
ARI6H5YR	6H Average Recurrance Interval Accumulation 5 Year
PRP06H	6 hour Precipitation Accumulation Return Period
GaugeInfIndex06H	6 hour QPE Gauge Influence Index
QPEFFG06H	6 hour QPE-to-FFG Ratio
FFG06	6-hr flash flood guidance
FFR06	6-hr flash flood runoff values
QPE06	6-hr Quantitative Precip Estimate
QPE06_ACR	6-hr Quantitative Precip Estimate
QPE06_ALR	6-hr Quantitative Precip Estimate
QPE06_FWR	6-hr Quantitative Precip Estimate
QPE06_KRF	6-hr Quantitative Precip Estimate
QPE06_MSR	6-hr Quantitative Precip Estimate
QPE06_ORN	6-hr Quantitative Precip Estimate
QPE06_PTR	6-hr Quantitative Precip Estimate
QPE06_RHA	6-hr Quantitative Precip Estimate
QPE06_RSA	6-hr Quantitative Precip Estimate
QPE06_STR	6-hr Quantitative Precip Estimate
QPE06_TAR	6-hr Quantitative Precip Estimate
QPE06_TIR	6-hr Quantitative Precip Estimate
QPE06_TUA	6-hr Quantitative Precip Estimate
QPF06	6-hr Quantitative Precip Forecast
QPF06_ACR	6-hr Quantitative Precip Forecast
QPF06_ALR	6-hr Quantitative Precip Forecast
QPF06_FWR	6-hr Quantitative Precip Forecast
QPF06_KRF	6-hr Quantitative Precip Forecast
QPF06_MSR	6-hr Quantitative Precip Forecast
QPF06_ORN	6-hr Quantitative Precip Forecast
QPF06_PTR	6-hr Quantitative Precip Forecast
QPF06_RHA	6-hr Quantitative Precip Forecast
QPF06_RSA	6-hr Quantitative Precip Forecast
QPF06_STR	6-hr Quantitative Precip Forecast
QPF06_TAR	6-hr Quantitative Precip Forecast
QPF06_TIR	6-hr Quantitative Precip Forecast

Table 1 – continued from previo

Abbreviation	Description
QPF06_TUA	6-hr Quantitative Precip Forecast
TCSRG70	70% Tropical Cyclone Storm Surge Exceedance
GaugeInfIndex72H	72 hour QPE Gauge Influence Index
TCSRG80	80% Tropical Cyclone Storm Surge Exceedance
TCSRG90	90% Tropical Cyclone Storm Surge Exceedance
ABSD	Absolute divergence
ABSH	Absolute Humidity
AH	Absolute humidity
ABSV	Absolute vorticity
ASD	Accumulated Snow Depth
ACOND	Aerodynamic conductance
AETYP	Aerosol type
AC137	Air concentration of Caesium 137
AI131	Air concentration of Iodine 131
ARADP	Air concentration of radioactive pollutant
ALBDO	Albedo
ACWVH	Altimeter corrected wave height
ALRRC	Altimeter Range Relative Correction
ASET	Altimeter setting
AWVH	Altimeter wave height
ALTMSL	Altitude above mean sea level
ANCConvectiveOutlook	ANC Convective Outlook
ANCFinalForecast	ANC Final Forecast
AOSGSO	Angle Of Sub-Grid Scale Orography
ASGSO	Anisotropy Of Sub-Grid Scale Orography
APTMP	Apparent Temperature
ARBTXT	Arbitrary text string
ASHFL	Assimilative Heat Flux
AMIXL	Asymptotic mixing length scale
ATMDIV	Atmospheric Divergence
AVSFT	Average surface skin temperature
BARET	Bare soil surface skin temperature
BKENG	Barotropic Kinectic Energy
UBARO	Barotropic U velocity
UBARO	Barotropic U Velocity
VBARO	Barotropic V velocity
VBARO	Barotropic V Velocity
BGRUN	Baseflow-groundwater runoff
BASRV	Base radial velocity
BASR	Base reflectivity
BASSW	Base spectrum width
4LFTX	Best (4-layer) lifted index
4LFTX	Best (4 layer) Lifted Index
BLI	Best lifted index (to 500 mb)
BMIXL	Blackadars mixing length scale
BLST	Bottom layer soil temperature
NONE	Bottom of Ocean Isothermal Layer
OBIL	Bottom of Ocean Isothermal Layer
NONE	Bottom of Ocean Mixed Layer (m)

Table 1 – continued from previo

Abbreviation	Description
OBML	Bottom of Ocean Mixed Layer (m)
BCBL	Boundary layer cloud bottom level
BCBL	Boundary Layer Cloud Bottom Level
BCLY	Boundary Layer Cloud Layer
BCY	Boundary layer cloud layer
BCY	Boundary Layer Cloud Layer
BCTL	Boundary layer cloud top level
BCTL	Boundary Layer Cloud Top Level
BLYSP	Boundary layer dissipation
BrightBandBottomHeight	Bright Band Bottom Height
BrightBandTopHeight	Bright Band Top Height
BRTMP	Brightness temperature
CAIIRAD	CaII-K Radiance
CCOND	Canopy conductance
EVCW	Canopy water evaporation
CONVP	Categorical Convection
CFRZR	Categorical Freezing Rain
CFRZR	Categorical Freezing Rain
CFRZR	Categorical Freezing Rain
CFRZR	Categorical freezing rain
CFRZR	Categorical Freezing Rain (yes=1; no=0)
CFRZR	Categorical Freezing Rain (yes=1; no=0)
CICEP	Categorical Ice Pellets
CICEP	Categorical Ice Pellets
CICEP	Categorical Ice Pellets
CICEP	Categorical ice pellets
CICEP	Categorical Ice Pellets (yes=1; no=0)
CICEP	Categorical Ice Pellets (yes=1; no=0)
CLGTN	Categorical Lightning
OZCAT	Categorical Ozone Concentration
OZCAT	Categorical Ozone Concentration
CRAIN	Categorical Rain
CRAIN	Categorical Rain
CRAIN	Categorical Rain
CRAIN	Categorical rain
CRAIN	Categorical Rain (yes=1; no=0)
CRAIN	Categorical Rain (yes=1; no=0)
SVRTS	Categorical Servre Thunderstorm
SVRTS	Categorical Severe Thunderstorm
CSNOW	Categorical Snow
CSNOW	Categorical Snow
CSNOW	Categorical Snow
CSNOW	Categorical snow
CSNOW	Categorical Snow (yes=1; no=0)
CSNOW	Categorical Snow (yes=1; no=0)
CTSTM	Categorical Thunderstorm (1-yes, 0-no)
TSTMC	Categorical Thunderstorm (1-yes, 0-no)
CCEIL	Ceiling
LightningDensity15min	CG Lightning Density (15 min.)

Table 1 – continued from previo

Abbreviation	Description
LightningDensity1min	CG Lightning Density (1 min.)
LightningDensity30min	CG Lightning Density (30 min.)
LightningDensity5min	CG Lightning Density (5 min.)
LightningProbabilityNext30min	CG Lightning Probability (0-30 min.)
CAT	Clear Air Turbulence (CAT)
CAT	Clear Air Turbulence(CAT)
CSDLF	Clear Sky Downward Long Wave Flux
CSDSF	Clear sky downward solar flux
CSULF	Clear Sky Upward Long Wave Flux
CSUSF	Clear Sky Upward Solar Flux
CDUVB	Clear sky UV-B downward solar flux
CAMT	Cloud amount
CBL	Cloud Base Level
CBASE	Cloud base
CEIL	Cloud Ceiling
CLG	Cloud ceiling
CLG	Cloud Ceiling
CloudCover	Cloud Cover
CFNLF	Cloud Forcing Net Long Wave Flux
CFNSF	Cloud Forcing Net Solar Flux
CDCIMR	Cloud Ice Mixing Ratio
CICE	Cloud ice
CLOUDM	Cloud mask
CLWMR	Cloud Mixing Ratio
CLWMR	Cloud mixing ratio
CTOPHQI	Cloud top height quality indicator
CTOP	Cloud top
heightCTHGT	Cloud top
CTYP	Cloud type
CWAT	Cloud water
CWORK	Cloud work function
CWORK	Cloud Work Function
CDWW	Coefficient of Drag With Waves
CISOILW	Column-Integrated Soil Water
REFC	Composite reflectivity
MergedReflectivityQCComposite	Composite Reflectivity
HeightCompositeReflectivity	Composite Reflectivity Height
MergedReflectivityQComposite	Composite Reflectivity Mosaic
EF25M	Cond 25% pcpn smy fractile past 24 hrs
EF50M	Cond 50% pcpn smy fractile past 24 hrs
TCOND	Condensate
CONDE	Condensate
CONDP	Condensation Pressure of Parcali Lifted From Indicate Surface
CONDP	Condensation Pressure of Parcal Lifted From Indicate Surface
CPPAF	Conditional percent precipitation amount fractile for an overall period (Encoded as an accumu
CICEL	Confidence - Ceiling
CIFLT	Confidence - Flight Category
CIVIS	Confidence - Visibility
CONTB	Contrail base

Table 1 – continued from previo

Abbreviation	Description
CONTB	Contrail Base
CONTE	Contrail engine type
CONTET	Contrail Engine Type
CONTI	Contrail intensity
CONTI	Contrail Intensity
CONTT	Contrail top
CONTT	Contrail Top
CONUSMergedReflectivity	CONUS Merged Reflectivity
CONUSPlusMergedReflectivity	CONUS-Plus Merged Reflectivity
CONVP	Convection Potential
CAPE	Convective available potential energy
CCBL	Convective cloud bottom level
CCBL	Convective Cloud Bottom Level
CCLY	Convective Cloud
CCY	Convective cloud
CCY	Convective Cloud
CDCON	Convective cloud cover
CUEFI	Convective Cloud Efficiency
CUEFI	Convective Cloud Efficiency
CUEFI	Convective cloud efficiency
MFLUX	Convective Cloud Mass Flux
CCTL	Convective cloud top level
CCTL	Convective Cloud Top Level
CNVDEMF	Convective detrainment mass flux
CNVDMF	Convective downdraft mass flux
CNGWDV	Convective Gravity wave drag meridional acceleration
CNGWDU	Convective Gravity wave drag zonal acceleration
CIN	Convective inhibition
CNVV	Convective meridional momentum mixing acceleration
ACPCP	Convective Precipitation
ACPCP	Convective precipitation
ACPCPN	Convective precipitation (nearest grid point)
ACPCPN	Convective precipitation (nearest grid point)
CPRAT	Convective Precipitation Rate
CPRAT	Convective Precipitation Rate
CPRAT	Convective precipitation rate
CPRAT	Convective Precipitation Rate
CSRATE	Convective Snowfall Rate
CSRATE	Convective Snowfall Rate
CSRWE	Convective Snowfall Rate Water Equivalent
CSRWE	Convective Snowfall Rate Water Equivalent
SNOC	Convective snow
CNVUMF	Convective updraft mass flux
CWP	Convective Water Precipitation
CWP	Convective Water Precipitation
CWDI	Convective Weather Detection Index
CNVU	Convective zonal momentum mixing acceleration
SNO C	Convect Snow
NTRNFLUX	Cosmic Ray Neutron Flux

Table 1 – continued from previo

Abbreviation	Description
COVTZ	Covariance between izonal component of the wind and temperature. Defined as [uT]-[u][T], v
COVMM	Covariance between meridional and meridional components of the wind. Defined as [vv]-[v][
COVMZ	Covariance between Meridional and Zonal Components of the wind.
COVTM	Covariance between meridional component of the wind and temperature. Defined as [vT]-[v]
COVQM	Covariance between specific humidity and meridional components of the wind. Defined as [v
COVQQ	Covariance between specific humidy and specific humidy. Defined as [qq]-[q][q], where "[]"
COVQVV	Covariance between specific humidity and vertical components of the wind. Defined as [Ωq]-
COVQZ	Covariance between specific humidity and zonal components of the wind. Defined as [uq]-[u]
COVPSPS	Covariance between surface pressure and surface pressure. Defined as [Psfc]-[Psfc][Psfc], wh
COVTM	Covariance between Temperature and Meridional Components of the wind.
COVTT	Covariance between temperature and temperature. Defined as [TT]-[T][T], where "[]" indicat
COVTW	Covariance between temperature and vertical component of the wind. Defined as [wT]-[w][T
COVTVV	Covariance between temperature and vertical components of the wind. Defined as [ΩT]-[Ω][T
COVTZ	Covariance between Temperature and Zonal Components of the wind.
COVVVVV	Covariance between vertical and vertical components of the wind. Defined as [ΩΩ]-[Ω][Ω], w
COVMZ	Covariance between zonal and meridional components of the wind. Defined as [uv]-[u][v], wh
COVZZ	Covariance between zonal and zonal components of the wind. Defined as [uu]-[u][u], where "
CrestMaxStreamflow	CREST Maximum Streamflow
CrestMaxUStreamflow	CREST Maximum Unit Streamflow
CrestSoilMoisture	CREST Soil Moisture
CRTFRQ	Critical Frequency
CB	Cumulonimbus Base
CBHE	Cumulonimbus Horizontal Exten
CT	Cumulonimbus Top
DIRC	Current direction
SPC	Current speed
DCBL	Deep convective cloud bottom level
DCBL	Deep Convective Cloud Bottom Level
DCCBL	Deep Convective Cloud Bottom Level
DCCTL	Deep Convective Cloud Top Level
DCTL	Deep convective cloud top level
DCTL	Deep Convective Cloud Top Level
CNVHR	Deep Convective Heating rate
CNVMR	Deep Convective Moistening Rate
CNVMR	Deep Convective Moistening Rate
DALT	Density altitude
DEN	Density
DBLL	Depth Below Land surface
DPBLW	Depth below land surface
DBSL	Depth Below Sea Level
DPMSL	Depth below sea level
REFZI	Derived radar reflectivity backscatter from ice
REFZI	Derived radar reflectivity backscatter from ice
REFZC	Derived radar reflectivity backscatter from parameterized convection
REFZC	Derived radar reflectivity backscatter from parameterized convection
REFZR	Derived radar reflectivity backscatter from rain
REFZR	Derived radar reflectivity backscatter from rain
REFD	Derived radar reflectivity
DEVMSL	Deviation of sea level from mean

Table 1 – continued from previo

Abbreviation	Description
DEPR	Dew point depression or deficit
DPT	Dew point temperature
DIREC	Direct Evaporation Cease(Soil Moisture)
SMDRY	Direct evaporation cease (soil moisture)
EVBS	Direct evaporation from bare soil
DIRWWW	Directional Width of The Wind Waves
Degree true	Direction Degrees true
WWSDIR	Direction of combined wind waves and swell
DICED	Direction of ice drift
SWDIR	Direction of swell waves
WVDIR	Direction of wind waves
DSKDAY	Disk Intensity Day
DSKINT	Disk Intensity
DSKNGT	Disk Intensity Night
DLWRF	Downward Long-Wave Rad. Flux
DLWRF	Downward long-wave radiation flux Downward Long-W/m^2
DSWRF	Downward Short-Wave Rad. Flux
DSWRF	Downward short-wave radiation flux Downward Short-W/m^2
DTRF	Downward Total radiation Flux
DWUVR	Downward UV Radiation
CD	Drag Coefficient
CD	Drag coefficient
ELON	East Longitude (0 - 360)
ELON	East Longitude (0 - 360)
ELONN	East Longitude (nearest neighbor) (0 - 360)
RETOP	Echo Top
RADT	Effective radiative skin temperature
ETOT	Electric Field Magnitude
ELCDEN	Electron Density
DIFEFLUX	Electron Flux (Differential)
INTEFLUX	Electron Flux (Integral)
ELECTMP	Electron Temperature
ELSCT	Elevation of snow covered terrain
EHELX	Energy helicity index
EATM	Entire atmosphere (considered as a single layer)
EA	Entire Atmosphere
EATM	Entire Atmosphere
NONE	Entire Atmosphere
EOCN	Entire ocean (considered as a single layer)
EOCN	Entire Ocean
NONE	Entire Ocean
EHLT	Equilibrium level
EHLT	Equilibrium Level
REFZC	Equivalent radar reflectivity factor for parameterized convection
REFZR	Equivalent radar reflectivity factor for rain
REFZI	Equivalent radar reflectivity factor for snow
ESTPC	Estimated precipitation
ESTUWIND	Estimated u component of wind
ESTVWIND	Estimated v component of wind

Table 1 – continued from previo

Abbreviation	Description
ELYR	Eta Level
ETAL	Eta Level
EUVRAD	EUV Radiance
EVP	Evaporation
EVP	Evaporation Evaporation - Precipitation
EMNP	Evaporation - Precipitation
EMNP	Evaporation - Precipitation
EVAPT	Evapotranspiration
SFEXC	Exchange coefficient
SFEXC	Exchange coefficient
ETCWL	Extra Tropical Storm Surge Combined Surge and Tide
ETSRG	Extra Tropical Storm Surge
F107	F10.7
FLDCP	Field Capacity
FIREDI	Fire Detection Indicator
FIREODT	Fire Outlook Due to Dry Thunderstorm
FIREOLK	Fire Outlook
FFG	Flash flood guidance (Encoded as an accumulation over a floating subinterval of time betwee
FFRUN	Flash flood runoff (Encoded as an accumulation over a floating subinterval of time)
FLGHT	Flight Category
QREC	Flood plain recharge
ModelHeight0C	Freezing Level Height
FRZR	Freezing Rain
FRZR	Freezing Rain
FPRATE	Freezing Rain Precipitation Rate
FPRATE	Freezing Rain Precipitation Rate
FREQ	Frequency
FRICV	Frictional velocity
FRICV	Frictional Velocity
FRICV	Friction Velocity
FROZR	Frozen Rain
FROZR	Frozen Rain
GHT	Geometrical height
DBSS	Geometric Depth Below Sea Surface
DIST	Geometric height
GPA	Geopotential height anomaly
GH	Geopotential height
HGT	Geopotential height
HGTN	Geopotential Height (nearest grid point)
GP	Geopotential
GRAD	Global radiation flux
GRAUP	Graupel (snow pellets)
GRLE	Grauple
GSGSO	Gravity Of Sub-Grid Scale Orography
GWDV	Gravity wave drag meridional acceleration
GWDU	Gravity wave drag zonal acceleration
GCBL	Grid scale cloud bottom level
GCBL	Grid Scale Cloud Bottom Level
GSCBL	Grid Scale Cloud Bottom Level

Table 1 – continued from previo

Abbreviation	Description
GCTL	Grid scale cloud top level
GCTL	Grid Scale Cloud Top Level
GSCTL	Grid Scale Cloud Top Level
GC137	Ground deposition of Caesium 137
GI131	Ground deposition of Iodine 131
GRADP	Ground deposition of radioactive
GFLUX	Ground heat flux
SFC	Ground or Water Surface
GWREC	Groundwater recharge
HAIL	Hail
HAILPROB	Hail probability
HINDEX	Haines Index
SIGHAL	Hall Conductivity
HARAD	H-Alpha Radiance
HFLUX	Heat Flux
HTX	Heat index
DIFIFLUX	Heavy Ion Flux (Differential)
INTIFLUX	Heavy Ion Flux (iIntegral)
HGTAG	Height above ground (see Note 1)
H50Above0C	Height of 50 dBZ Echo Above 0C
H50AboveM20C	Height of 50 dBZ Echo Above -20C
H60Above0C	Height of 60 dBZ Echo Above 0C
H60AboveM20C	Height of 60 dBZ Echo Above -20C
HELCOR	Heliospheric Radiance
ABSRB	HF Absorption
ABSFRQ	HF Absorption Frequency
HPRIMF	h'F
HCBL	High cloud bottom level
HCBL	High Cloud Bottom Level
HCDC	High cloud cover
HCL	High cloud layer
HCL	High Cloud Layer
HCLY	High Cloud Layer
HCTL	High cloud top level
HCTL	High Cloud Top Level
HSCLW	Highest top level of supercooled liquid water layer
HSCLW	Highest Top Level of Supercooled Liquid Water Layer
HTSLW	Highest Top Level of Supercooled Liquid Water Layer
HTFL	Highest tropospheric freezing level
HTFL	Highest Tropospheric Freezing Level
HighLayerCompositeReflectivity	High Layer Composite Reflectivity (24-60 kft)
HAVNI	High-Level aviation interest
HRCONO	High risk convective outlook
MCONV	Horizontal Moisture Convergence
HMC	Horizontal moisture convergence
MCONV	Horizontal Moisture Divergence
MCONV	Horizontal Moisture Divergence
MCONV	Horizontal moisture divergence
MCONV	Horizontal Moisture Divergence

Table 1 – continued from previo

Abbreviation	Description
MFLX	Horizontal momentum flux
MFLX	Horizontal Momentum Flux
MFLX	Horizontal Momentum Flux
CompositeReflectivityMaxHourly	Hourly Composite Reflectivity Maximum
MAXDVV	Hourly Maximum of Downward Vertical Vorticity in the lowest 400hPa
MAXREF	Hourly Maximum of Simulated Reflectivity at 1 km AGL
MXUPHL	Hourly Maximum of Updraft Helicity over Layer 2-5 km AGL
MXUPHL	Hourly Maximum of Updraft Helicity over Layer 2km to 5 km AGL
MAXUVV	Hourly Maximum of Upward Vertical Vorticity in the lowest 400hPa
MIXR	Humidity Mixing Ratio
MIXR	Humidity mixing ratio
RCQ	Humidity parameterin canopy conductance
RCQ	Humidity parameter in canopy conductance
HYBL	Hybrid Level
HCBB	ICAO Height at Cumulonimbus Bas
HCBT	ICAO Height at Cumulonimbus To
HECBB	ICAO Height at Embedded Cumulonimbus Bas
HECBT	ICAO Height at Embedded Cumulonimbus To
ICAHT	ICAO Standard Atmosphere Reference Height
ICEC	Ice cover
ICED	Ice divergence
FICE	Ice fraction of total condensate
FICE	Ice fraction of total condensate
FICE	Ice fraction of total condensate
surface	Ice-free water
ICWAT	Ice-free water surface
ICEG	Ice growth rate
IPRATE	Ice Pellets Precipitation Rate
IPRATE	Ice Pellets Precipitation Rate
ICE T	Ice Temperature
ICETK	Ice thickness
ICMR	Ice Water Mixing Ratio
ICMR	Ice water mixing ratio
ICIB	Icing base
ICIB	Icing Base
ICIP	Icing
ICIP	Icing Potential
TIPD	Icing Potential
ICPRB	Icing probability
ICI	Icing Icing Severity
ICSEV	Icing severity
ICIT	Icing top
ICIT	Icing Top
CTP	In-Cloud Turbulence
IRBand4	Infrared Imagery
INSTRR	Instantaneous rain rate
LIPMF	Integrated column particulate matter (fine)
LIPMF	Integrated column particulate matter (fine)
ILIQW	Integrated Liquid Water

Table 1 – continued from previo

Abbreviation	Description
ILW	Integrated liquid water
TSI	Integrated Solar Irradiance
INTFD	Interface Depths
IMFTSW	Inverse Mean Frequency of The Total Swell
IMFWW	Inverse Mean Frequency of The Wind Waves
IMWF	Inverse Mean Wave Frequency
IONDEN	Ion Density
IDRL	Ionospheric D-region level
IERL	Ionospheric E-region level
IF1RL	Ionospheric F1-region level
IF2RL	Ionospheric F2-region level
IONTMP	Ion Temperature
THEL	Isentropic (theta) level
ISBL	Isobaric Surface
TMPL	Isothermal Level
KX	K index
KENG	Kinetic Energy
MELBRNE	KNES1
KOX	KO index
BENINX	Kurtosis of The Sea Surface Elevation Due to Waves
LAND	Land cover (0=sea, 1=land)
LANDN	Land-sea coverage (nearest neighbor)
LSPA	Land Surface Precipitation Accumulation
LANDU	Land use
LAPR	Lapse rate
LRGHR	Large Scale Condensate Heating rate
LRGMR	Large scale moistening rate
NCPCP	Large-Scale Precipitation (non-convective)
NCPCP	Large scale precipitation (non-convective)
LSPRATE	Large Scale Precipitation Rate
LSPRATE	Large Scale Precipitation Rate
LSSRATE	Large Scale Snowfall Rate
LSSRATE	Large Scale Snowfall Rate
LSSRWE	Large Scale Snowfall Rate Water Equivalent
LSSRWE	Large Scale Snowfall Rate Water Equivalent
SNO L	Large-Scale Snow
SNOL	Large scale snow
LSWP	Large Scale Water Precipitation (Non-Convective)
LSWP	Large Scale Water Precipitation (Non-Convective)
LHTFL	Latent heat net flux
NLAT	Latitude (-90 to +90)
NLAT	Latitude (-90 to +90)
NLATN	Latitude (nearest neighbor) (-90 to +90)
LAPP	Latitude of Presure Point
LAUV	Latitude of U Wind Component of Velocity
LAVV	Latitude of V Wind Component of Velocity
NONE	Layer Between Two Depths Below Ocean Surface
OLYR	Layer between two depths below ocean surface
OLYR	Layer Between Two Depths Below Ocean Surface

Table 1 – continued from previo

Abbreviation	Description
LBTHL	Layer Between Two Hybrid Levels
NONE	Layer Between Two Hybrid Levels
LMBSR	Layer-maximum base reflectivity
LOS	Layer Ocean Surface and 26C Ocean Isothermal Level
NONE	Layer Ocean Surface and 26C Ocean Isothermal Level
LAYTH	Layer Thickness
LAI	Leaf Area Index
PDLY	Level at Specified Pressure Difference from Ground to Level
SPDL	Level at Specified Pressure Difference from Ground to Level
0DEG	level of 0C Isotherm
ADCL	Level of Adiabatic Condensation Lifted from the Surface
CTL	Level of Cloud Tops
LTNG	Lightning
LTNG	Lightning
LMBINT	Limb Intensity
ARAIN	Liquid precipitation (rainfall)
ARAIN	Liquid precipitation (rainfall)
LSOIL	Liquid soil moisture content (non-frozen)
LIQVSM	Liquid Volumetric Soil Moisture (Non-Frozen)
SOILL	Liquid volumetric soil moisture (non-frozen)
LOPP	Longitude of Presure Point
LOUV	Longitude of U Wind Component of Velocity
LOVV	Longitude of V Wind Component of Velocity
LWAVR	Long wave radiation flux
LWHR	Long-Wave Radiative Heating Rate
LCBL	Low cloud bottom level
LCBL	Low Cloud Bottom Level
LCDC	Low cloud cover
LCLY	Low Cloud Layer
LCY	Low cloud layer
LCY	Low Cloud Layer
LCTL	Low cloud top level
LCTL	Low Cloud Top Level
LLSM	Lower layer soil moisture
LBSLW	Lowest Bottom Level of Supercooled Liquid Water Layer
LSCLW	Lowest bottom level of supercooled liquid water layer
LSCLW	Lowest Bottom Level of Supercooled Liquid Water Layer
LLTW	Lowest level of the wet bulb zero
LLTW	Lowest Level of the Wet Bulb Zero
LWBZ	Lowest Level of the Wet Bulb Zero
WBZ	Lowest Level of the Wet Bulb Zero
LowLayerCompositeReflectivity	Low Layer Composite Reflectivity (0-24 kft)
LAVNI	Low-Level aviation interest
MergedAzShear02kmAGL	Low-Level Azimuthal Shear (0-2km AGL)
LLCompositeReflectivity	Low-Level Composite Reflectivity
HeightLLCompositeReflectivity	Low-Level Composite Reflectivity Height
RotationTrackLL120min	Low-Level Rotation Tracks 0-2km AGL (120 min. accum.)
RotationTrackLL1440min	Low-Level Rotation Tracks 0-2km AGL (1440 min. accum.)
RotationTrackLL240min	Low-Level Rotation Tracks 0-2km AGL (240 min. accum.)

Table 1 – continued from previo

Abbreviation	Description
RotationTrackLL30min	Low-Level Rotation Tracks 0-2km AGL (30 min. accum.)
RotationTrackLL360min	Low-Level Rotation Tracks 0-2km AGL (360 min. accum.)
RotationTrackLL60min	Low-Level Rotation Tracks 0-2km AGL (60 min. accum.)
BTOT	Magnetic Field Magnitude
MTHA	Main thermocline anomaly
MTHD	Main thermocline depth
LMH	Mass Point Model Surface
MAXAH	Maximum absolute humidity
MAXAH	Maximum Absolute Humidity
MACAT	Maximum Cloud Air Turbulence Potential
REFC	Maximum/Composite radar reflectivity
MTHE	Maximum equivalent potential temperature level
MTHE	Maximum Equivalent Potential Temperature level
MESH	Maximum Estimated Size of Hail (MESH)
MAIP	Maximum Icing Potential
MAXWH	Maximum Individual Wave Height
PRPMax	Maximum Precipitation Return Period
QPEFFGMax	Maximum QPE-to-FFG Ratio
MAXRH	Maximum relative humidity
MAXRH	Maximum Relative Humidity
MXSALB	Maximum snow albedo
MXSALB	Maximum Snow Albedo
MXSALB	Maximum Snow Albedo*
QMAX	Maximum specific humidity at 2m
TMAX	Maximum temperature
MWSL	Maximum Wind Level
MAXWS	Maximum wind speed
MACTP	Max in-Cloud Turbulence Potential
MECAT	Mean Cloud Air Turbulence Potential
MEI	Mean Icing Potential
MECTP	Mean in-Cloud Turbulence Potential
MWSPER	Mean period of combined wind waves and swell
SWPER	Mean period of swell waves
WVPER	Mean period of wind waves
MSL	Mean Sea Level
M2SPW	Mean square slope of waves
MZPTSW	Mean Zero-Crossing Period of The Total Swell
MZPWW	Mean Zero-Crossing Period of The Wind Waves
MZWPER	Mean Zero-Crossing Wave Period
MCDC	Medium cloud cover
MergedBaseReflectivityQC	Merged Base Reflectivity
MergedReflectivityAtLowestAltitude	Merged Reflectivity At Lowest Altitude (RALA)
VGWD	Meridional flux of gravity wave stress
V-GWD	Meridional Flux of Gravity Wave Stress
MESHTrack120min	MESH Tracks (120 min. accum.)
MESHTrack1440min	MESH Tracks (1440 min. accum.)
MESHTrack240min	MESH Tracks (240 min. accum.)
MESHTrack30min	MESH Tracks (30 min. accum.)
MESHTrack360min	MESH Tracks (360 min. accum.)

Table 1 – continued from previo

Abbreviation	Description
MESHTrack60min	MESH Tracks (60 min. accum.)
MCBL	Middle cloud bottom level
MCBL	Middle Cloud Bottom Level
MCLY	Middle Cloud Layer
MCY	Middle cloud layer
MCY	Middle Cloud Layer
MCTL	Middle cloud top level
MCTL	Middle Cloud Top Level
MergedAzShear36kmAGL	Mid-Level Azimuthal Shear (3-6km AGL)
RotationTrackML120min	Mid-Level Rotation Tracks 3-6km AGL (120 min. accum.)
RotationTrackML1440min	Mid-Level Rotation Tracks 3-6km AGL (1440 min. accum.)
RotationTrackML240min	Mid-Level Rotation Tracks 3-6km AGL (240 min. accum.)
RotationTrackML30min	Mid-Level Rotation Tracks 3-6km AGL (30 min. accum.)
RotationTrackML360min	Mid-Level Rotation Tracks 3-6km AGL (360 min. accum.)
RotationTrackML60min	Mid-Level Rotation Tracks 3-6km AGL (60 min. accum.)
RSMIN	Minimal stomatal resistance
DEPMN	Minimum dew point depression
MINRH	Minimum Relative Humidity
QMIN	Minimum specific humidity at 2m
TMIN	Minimum temperature
Entire Atmosphere	Missing200
MIXHT	mixed layer depth
MIXHT	Mixed Layer Depth
MIXL	Mixed Layer Depth
MLYNO	Model Layer number (From bottom up)
MTHT	Model terrain height
MRCONO	Moderate risk convective outlook
MSTAV	Moisture availability
UFLX	Momentum flux, u component
VFLX	Momentum flux, v component
MNTSF	Montgomery stream function
MSLET	MSLP (Eta model reduction)
MSLPM	MSLP (MAPS System Reduction)
NLGSP	Natural Log of Surface Pressure
NBDSF	Near IR Beam Downward Solar Flux
NBSALB	Near IR, Black Sky Albedo
NDDSF	Near IR Diffuse Downward Solar Flux
NWSALB	Near IR, White Sky Albedo
AOHFLX	Net Air-Ocean Heat Flux
NLWRCS	Net Long-Wave Radiation Flux, Clear Sky
NLWRS	Net long wave radiation flux (surface)
NLWRT	Net long wave radiation flux (top of atmosphere)
NLWRF	Net Long-Wave Radiation Flux
NSWRFCS	Net Short-Wave Radiation Flux, Clear Sky
NSWRS	Net short-wave radiation flux (surface)
NSWRT	Net short-wave radiation flux (top of atmosphere)
NSWRF	Net Short Wave Radiation Flux
NTAT	Nominal Top of the Atmosphere
CDLYR	Non-convective cloud cover

Table 1 – continued from previo

Abbreviation	Description
CDLYR	Non-Convective Cloud Cover non-dim
NWSTR	Normalised Waves Stress
NDVI	Normalized Difference Vegetation Index
NCIP	Number concentration for ice particles
NCIP	Number concentration for ice particles
MIXLY	Number of mixed layers next to surface
NPIXU	Number Of Pixels Used
RLYRS	Number of soil layers in root zone
RLYRS	Number of soil layers in root zone
RLYRS	Number of soil layers in root zone
OHC	Ocean Heat Content
OITL	Ocean Isotherm Level (1/10 deg C)
NONE	Ocean Isotherm Level
OITL	Ocean Isotherm Level
NONE	Ocean Mixed Layer
OML	Ocean Mixed Layer
P2OMLT	Ocean Mixed Layer Potential Density (Reference 2000m)
OMLU	Ocean Mixed Layer U Velocity
OMLV	Ocean Mixed Layer V Velocity
ELEV	Ocean Surface Elevation Relative to Geoid
OMGALF	Omega (Dp/Dt) divide by density
EWATR	Open water evaporation (standing water)
OSD	Ordered Sequence of Data
OSEQ	Ordered Sequence of Data
OZCON	Ozone Concentration (PPB)
OZMAX1	Ozone Daily Max from 1-hour Average
OZMAX8	Ozone Daily Max from 8-hour Average
O3MR	Ozone Mixing Ratio
O3MR	Ozone mixing ratio
O3MR	Ozone Mixing Ratio
POZO	Ozone production from col ozone term
POZT	Ozone production from temperature term
POZ	Ozone production
TOZ	Ozone tendency
VDFOZ	Ozone vertical diffusion
SIGPAR	Parallel Conductivity
PRATMP	Parallel Temperature
PLI	Parcel lifted index (to 500 mb)
PLSMDEN	Particle Number Density
PMTC	Particulate matter (coarse)
PMTC	Particulate matter (coarse)
LPMTF	Particulate matter (fine)
LPMTF	Particulate matter (fine)
PMTF	Particulate matter (fine)
PMTF	Particulate matter (fine)
PPERTS	Peak Period of The Total Swell
PPERWW	Peak Period of The Wind Waves
PWPER	Peak Wave Period
SIGPED	Pedersen Conductivity

Table 1 – continued from previo

Abbreviation	Description
CPOFP	Percent frozen precipitation
PCTP1	Percent pcpn in 1st 6-h sub-period of 24 hr period
PCTP2	Percent pcpn in 2nd 6-h sub-period of 24 hr period
PCTP3	Percent pcpn in 3rd 6-h sub-period of 24 hr period
PCTP4	Percent pcpn in 4th 6-h sub-period of 24 hr period
PPSUB	Percent precipitation in a sub-period of an overall period (Encoded as per cent accumulation c
PMAXWH	Period of Maximum Individual Wave Height
PRPTMP	Perpendicular Temperature
PHOTAR	Photosynthetically Active Radiation
PIXST	Pixel scene type
BLD	Planetary Boundary Layer
HPBL	Planetary boundary layer height
HPBL	Planetary Boundary Layer Height
PLBL	Planetary Boundary Layer
PBLR	Planetary boundary layer regime
PBLREG	Planetary Boundary Layer Regime
CNWAT	Plant canopy surface water
PDMAX1	PM 2.5 Daily Max from 1-hour Average
PDMAX24	PM 2.5 Daily Max from 24-hour Average
PEVAP	Potential Evaporation
PEVAP	Potential evaporation
PEVAP	Potential Evaporation
PEVPR	Potential Evaporation Rage
PEVPR	Potential evaporation rate
PEVPR	Potential Evaporation Rate
THZ0	Potential temperature at top of viscous sublayer
POT	Potential temperature
POT	Potential temperature (theta)
PV	Potential vorticity
PV	Potential vorticity
PVL	Potential Vorticity
PVORT	Potential vorticity
PVMWW	Potential Vorticity (Mass-Weighted)
PWC	Precipitable water category
PWCAT	Precipitable Water Category
P WAT	Precipitable Water
PWAT	Precipitable water
PRCP	Precipitation
PRATE	Precipitation Rate
PRATE	Precipitation rate
PTYPE	Precipitation type
PTYPE	Precipitation Type
PR	Precip rate
(See note 3)	Predominant Weather
PWTHER	Predominant Weather
PALT	Pressure altitude
PRESA	Pressure anomaly
PCBB	Pressure at Cumulonimbus Bas
PCBT	Pressure at Cumulonimbus To

Table 1 – continued from previo

Abbreviation	Description
PECBB	Pressure at Embedded Cumulonimbus Bas
PECBT	Pressure at Embedded Cumulonimbus To
PRESDEV	Pressure deviation from ground to level
PRESD	Pressure deviation from mean sea level
PRESN	Pressure (nearest grid point)
PLPL	Pressure of level from which parcel was lifted
PLPL	Pressure of most parcel with highest theta-e in lowest 300 mb
P	Pressure
PRES	Pressure
PRES	Pressure
PRMSL	Pressure reduced to MSL
PTEND	Pressure tendency
DIRPW	Primary wave direction
PERPW	Primary wave mean period
POP	Probability of 0.01 inches of precipitation
POP	Probability of 0.01 inch of precipitation (POP)
PROCON	Probability of Convection
PPFFG	Probability of Excessive Rain
CPOZP	Probability of Freezing Precipitation
PFREZPREC	Probability of Freezing Precipitation
CPOFP	Probability of Frozen Precipitation
PFROZPREC	Probability of Frozen Precipitation
PPFFG	Probability of precipitation exceeding flash flood guidance values
POSH	Probability of Severe Hail (POSH)
WarmRainProbability	Probability of Warm Rain
CWR	Probability of Wetting Rain, exceeding in 0.10” in a given time period
PTAN	Prob of Temperature above normal
PTBN	Prob of Temperature below normal
PTNN	Prob of Temperature near normal
PPAN	Prob of Total Precipitation above normal
PPBN	Prob of Total Precipitation below normal
PPNN	Prob of Total Precipitation near normal
PROTDEN	Proton Density
DIFPFLUX	Proton Flux (Differential)
INTPFLUX	Proton Flux (Integral)
PROTTMP	Proton Temperature
EPOT	Pseudo-adiabatic potential temperature or equivalent potential temperature
MountainMapperQPE12H	QPE - Mountain Mapper (12 hr. accum.)
MountainMapperQPE01H	QPE - Mountain Mapper (1 hr. accum.)
MountainMapperQPE24H	QPE - Mountain Mapper (24 hr. accum.)
MountainMapperQPE03H	QPE - Mountain Mapper (3 hr. accum.)
MountainMapperQPE48H	QPE - Mountain Mapper (48 hr. accum.)
MountainMapperQPE06H	QPE - Mountain Mapper (6 hr. accum.)
MountainMapperQPE72H	QPE - Mountain Mapper (72 hr. accum.)
GaugeOnlyQPE12H	QPE - Radar Gauge Only (12 hr. accum.)
GaugeOnlyQPE01H	QPE - Radar Gauge Only (1 hr. accum.)
GaugeOnlyQPE24H	QPE - Radar Gauge Only (24 hr. accum.)
GaugeOnlyQPE03H	QPE - Radar Gauge Only (3 hr. accum.)
GaugeOnlyQPE48H	QPE - Radar Gauge Only (48 hr. accum.)

Table 1 – continued from previo

Abbreviation	Description
GaugeOnlyQPE06H	QPE - Radar Gauge Only (6 hr. accum.)
GaugeOnlyQPE72H	QPE - Radar Gauge Only (72 hr. accum.)
RadarOnlyQPE12H	QPE - Radar Only (12 hr. accum.)
RadarOnlyQPE01H	QPE - Radar Only (1 hr. accum.)
RadarOnlyQPE24H	QPE - Radar Only (24 hr. accum.)
RadarOnlyQPE03H	QPE - Radar Only (3 hr. accum.)
RadarOnlyQPE48H	QPE - Radar Only (48 hr. accum.)
RadarOnlyQPE06H	QPE - Radar Only (6 hr. accum.)
RadarOnlyQPE72H	QPE - Radar Only (72 hr. accum.)
GaugeCorrQPE12H	QPE - Radar with Gauge Bias Correction (12 hr. accum.)
GaugeCorrQPE01H	QPE - Radar with Gauge Bias Correction (1 hr. accum.)
GaugeCorrQPE24H	QPE - Radar with Gauge Bias Correction (24 hr. accum.)
GaugeCorrQPE03H	QPE - Radar with Gauge Bias Correction (3 hr. accum.)
GaugeCorrQPE48H	QPE - Radar with Gauge Bias Correction (48 hr. accum.)
GaugeCorrQPE06H	QPE - Radar with Gauge Bias Correction (6 hr. accum.)
GaugeCorrQPE72H	QPE - Radar with Gauge Bias Correction (72 hr. accum.)
PrecipRate	Radar Precipitation Rate (SPR)
RadarQualityIndex	Radar Quality Index (RQI)
RDSP1	Radar spectra (1)
RDSP2	Radar spectra (2)
RDSP3	Radar spectra (3)
RDLNUM	Radial number (2pi/lambda)
SWRAD	Radiance (with respect to wave length)
LWRAD	Radiance (with respect to wave number)
EPSR	Radiative emissivity
EPSR	Radiative emissivity
FRAIN	Rain fraction of total cloud water
FRAIN	Rain Fraction of Total Liquid Water
FRAIN	Rain Fraction of Total Liquid Water
FRAIN	Rain Fraction of Total Liquid Water
RWMR	Rain Mixing Ratio
RWMR	Rain mixing ratio
RPRATE	Rain Precipitation Rate
RPRATE	Rain Precipitation Rate
RDRIP	Rate of water dropping from canopy to ground
ground	Rate of water dropping from canopy to
MergedReflectivityComposite	Raw Composite Reflectivity Mosaic
MergedBaseReflectivity	Raw Merged Base Reflectivity
RFL06	Reflectance in 0.6 Micron Channel
RFL08	Reflectance in 0.8 Micron Channel
RFL16	Reflectance in 1.6 Micron Channel
RFL39	Reflectance in 3.9 Micron Channel
Reflectivity0C	Reflectivity at 0C
ReflectivityM10C	Reflectivity at -10C
ReflectivityM15C	Reflectivity at -15C
REFD	Reflectivity at 1 km AGL
ReflectivityM20C	Reflectivity at -20C
ReflectivityM5C	Reflectivity at -5C
ReflectivityAtLowestAltitude	Reflectivity At Lowest Altitude (RALA)

Table 1 – continued from previo

Abbreviation	Description
REFD	Reflectivity
RAZA	Relative Azimuth Angle
RELD	Relative divergence
REV	Relative Error Variance
RH	Relative humidity
R H	Relative Humidity
RHPW	Relative Humidity with Respect to Precipitable Water
RELV	Relative vorticity
RSSC	Remotely sensed snow cover
RI	Richardson number
RI	Richardson Number
FRIME	Rime Factor
RIME	Rime Factor
RIME	Rime factor
RIME	Rime Factor
RIME	Rime Factor
SFCRH	Roughness length for heat
SALTY	Salinity
SLTFL	Salt Flux
SATD	Saturation deficit
SAT D	Saturation Deficit
SATOSM	Saturation Of Soil Moisture
SCALB	Scaled albedo
SCBT	Scaled brightness temperature
SCCTP	Scaled cloud top pressure
SCLI	Scaled lifted index
SCPW	Scaled precipitable water
SCRAD	Scaled radiance
SCST	Scaled skin temperature
SCESTUWIND	Scatterometer Estimated U Wind Component
SCESTVWIND	Scatterometer Estimated V Wind Component
SCINT	Scintillation
SEAB	Sea Bottom
SeamlessHSR	Seamless Hybrid Scan Reflectivity (SHSR)
SeamlessHSRHeight	Seamless Hybrid Scan Reflectivity (SHSR) Height
SSHG	Sea Surface Height Relative to Geoid
DIRSW	Secondary wave direction
PERSW	Secondary wave mean periods
s	Seconds prior to initial reference time (defined in Section 1)
TSEC	Seconds prior to initial reference time
TSEC	Seconds Prior To Initial Reference Time
SHTFL	Sensible heat net flux
SHI	Severe Hail Index (SHI)
SCBL	Shallow convective cloud bottom level
SCBL	Shallow Convective Cloud Bottom Level
SCCBT	Shallow Convective Cloud Bottom Level
SCCTL	Shallow Convective Cloud Top Level
SCTL	Shallow convective cloud top level
SCTL	Shallow Convective Cloud Top Level

Table 1 – continued from previo

Abbreviation	Description
SHAHR	Shallow Convective Heating rate
SHAMR	Shallow Convective Moistening Rate
SHAMR	Shallow Convective Moistening Rate
SWAVR	Short wave radiation flux
SGCVV	Sigma coordinate vertical velocity
SIGL	Sigma Level
SHAILPRO	Significant Hail probability
SIGHAILPROB	Significant Hail probability
HTSGW	Significant height of combined wind waves and swell
SWELL	Significant height of swell waves
WVHGT	Significant height of wind waves (m)
SIGTRNDPROB	Significant Tornado probability
STORPROB	Significant Tornado probability
SIGWINDPROB	Significant Wind probability
SWINDPRO	Significant Wind probability Silt loam
SBC123	Simulated Brightness Counts for GOES 12, Channel 3
SBC124	Simulated Brightness Counts for GOES 12, Channel 4
SBTA1610	Simulated Brightness Temperature for ABI GOES-16, Band-10
SBTA1611	Simulated Brightness Temperature for ABI GOES-16, Band-11
SBTA1612	Simulated Brightness Temperature for ABI GOES-16, Band-12
SBTA1613	Simulated Brightness Temperature for ABI GOES-16, Band-13
SBTA1614	Simulated Brightness Temperature for ABI GOES-16, Band-14
SBTA1615	Simulated Brightness Temperature for ABI GOES-16, Band-15
SBTA1616	Simulated Brightness Temperature for ABI GOES-16, Band-16
SBTA167	Simulated Brightness Temperature for ABI GOES-16, Band-7
SBTA168	Simulated Brightness Temperature for ABI GOES-16, Band-8
SBTA169	Simulated Brightness Temperature for ABI GOES-16, Band-9
SBTA1710	Simulated Brightness Temperature for ABI GOES-17, Band-10
SBTA1711	Simulated Brightness Temperature for ABI GOES-17, Band-11
SBTA1712	Simulated Brightness Temperature for ABI GOES-17, Band-12
SBTA1713	Simulated Brightness Temperature for ABI GOES-17, Band-13
SBTA1714	Simulated Brightness Temperature for ABI GOES-17, Band-14
SBTA1715	Simulated Brightness Temperature for ABI GOES-17, Band-15
SBTA1716	Simulated Brightness Temperature for ABI GOES-17, Band-16
SBTA177	Simulated Brightness Temperature for ABI GOES-17, Band-7
SBTA178	Simulated Brightness Temperature for ABI GOES-17, Band-8
SBTA179	Simulated Brightness Temperature for ABI GOES-17, Band-9
AMSRE10	Simulated Brightness Temperature for AMSRE on Aqua, Channel 10
AMSRE11	Simulated Brightness Temperature for AMSRE on Aqua, Channel 11
AMSRE12	Simulated Brightness Temperature for AMSRE on Aqua, Channel 12
AMSRE9	Simulated Brightness Temperature for AMSRE on Aqua, Channel 9
SBT112	Simulated Brightness Temperature for GOES 11, Channel 2
SBT113	Simulated Brightness Temperature for GOES 11, Channel 3
SBT114	Simulated Brightness Temperature for GOES 11, Channel 4
SBT115	Simulated Brightness Temperature for GOES 11, Channel 5
SBT122	Simulated Brightness Temperature for GOES 12, Channel 2
SBT123	Simulated Brightness Temperature for GOES 12, Channel 3
SBT124	Simulated Brightness Temperature for GOES 12, Channel 4
SBT125	Simulated Brightness Temperature for GOES 12, Channel 5

Table 1 – continued from previo

Abbreviation	Description
SBT124	Simulated Brightness Temperature for GOES E Infrared
SBT123	Simulated Brightness Temperature for GOES E Water Vapor
SBT114	Simulated Brightness Temperature for GOES W Infrared
SBT113	Simulated Brightness Temperature for GOES W Water Vapor
SRFA161	Simulated Reflectance Factor for ABI GOES-16, Band-1
SRFA162	Simulated Reflectance Factor for ABI GOES-16, Band-2
SRFA163	Simulated Reflectance Factor for ABI GOES-16, Band-3
SRFA164	Simulated Reflectance Factor for ABI GOES-16, Band-4
SRFA165	Simulated Reflectance Factor for ABI GOES-16, Band-5
SRFA166	Simulated Reflectance Factor for ABI GOES-16, Band-6
SRFA171	Simulated Reflectance Factor for ABI GOES-17, Band-1
SRFA172	Simulated Reflectance Factor for ABI GOES-17, Band-2
SRFA173	Simulated Reflectance Factor for ABI GOES-17, Band-3
SRFA174	Simulated Reflectance Factor for ABI GOES-17, Band-4
SRFA175	Simulated Reflectance Factor for ABI GOES-17, Band-5
SRFA176	Simulated Reflectance Factor for ABI GOES-17, Band-6
SKTMP	Skin Temperature
SRCONO	Slight risk convective outlook
SSGSO	Slope Of Sub-Grid Scale Orography
SNOAG	Snow age
SNOAG	Snow Age
SALBD	Snow Albedo
SCE	Snow Cover by elevation (snow=0-252,neither=253,clouds=254)
SCP	Snow Cover
SC	Snow Cover (snow=250,clouds=100,neither=50)
SNOWC	Snow cover
SNOWC	Snow Cover
SDEN	Snow Density
SDEN	Snow Density
SNOD	Snow depth
SNO D	Snow Depth
SDWE	Snow Depth Water Equivalent
SDWE	Snow Depth Water Equivalent
SEVAP	Snow Evaporation
SEVAP	Snow Evaporation
SRWEQ	Snowfall Rate Water Equivalent
SRWEQ	Snowfall rate water equivalent
SNFALB	Snow free albedo
SNFALB	Snow-Free Albedo
SNO M	Snow Melt
SNOM	Snow melt
SNMR	Snow Mixing Ratio
SNMR	Snow mixing ratio
SNOHF	Snow phase change heat flux
SNOHF	Snow Phase Change Heat Flux
SPRATE	Snow Precipitation Rate
SPRATE	Snow Precipitation Rate
SNOWT	Snow temperature, depth-avg
SNOT	Snow temperature

Table 1 – continued from previo

Abbreviation	Description
SNO T	Snow temperature
SNOT	Snow Temperature
SCE	Snow water equivalent
SWEPN	Snow water equivalent percent of normal
SOILM	Soil moisture content
SOILM	Soil Moisture
RCSOL	Soil moisture parameter in canopy conductance
RCSOL	Soil moisture parameter in canopy conductance
SOILP	Soil Porosity
POROS	Soil porosity
TSOIL	Soil temperature
SOTYP	Soil type
EUVIRR	Solar EUV Irradiance
RCS	Solar parameter in canopy conductance
RCS	Solar parameter in canopy conductance
SP	Solar photosphere
SWHR	Solar Radiative Heating Rate
SOLRF	Solar Radio Emissions
SPECIRR	Solar Spectral Irradiance
XLONG	Solar X-ray Flux (XRS Long)
XSHRT	Solar X-ray Flux (XRS Short)
SOLZA	Solar Zenith Angle
AMSL	Specific Altitude Above Mean Sea Level
QZ0	Specific humidity at top of viscous sublayer
SPF H	Specific Humidity
SPFH	Specific humidity
HTGL	Specified Height Level Above Ground
SRCS	Specified radius from the center of the Sun
DWWW	Spectal directional width of the wind waves
SPFTR	Spectral Peakedness Factor
SICED	Speed of ice drift
SPRDF	Spread F
HSTDV	Standard deviation of height
SDSGSO	Standard Deviation Of Sub-Grid Scale Orography
TSD1D	Standard Dev. of IR Temp. over 1x1 deg. area
HLCY	Storm relative helicity
SSRUN	Storm surface runoff
SURGE	Storm Surge
STPA	Storm total precip accum
STRM	Stream function
SBSNO	Sublimation (evaporation from snow)
SBSNO	Sublimation (evaporation from snow)
SUN	Sunshine duration (ECMWF proposal, not WMO approved)
SUNSD	Sunshine Duration
SUNS	SunShine
SIPD	Supercooled Large Droplet Icing
SIPD	Supercooled Large Droplet (SLD) Icing
SLDP	Supercooled Large Droplet (SLD) Probabilitysee note 1
SLDP	Supercooled Large Droplet (SLD) Probability

Table 1 – continued from previo

Abbreviation	Description
SuperLayerCompositeReflectivity	Super Layer Composite Reflectivity (33-60 kft)
AKHS	Surface exchange coefficients for T and Q divided by delta z
AKMS	Surface exchange coefficients for U and V divided by delta z
LFTX	Surface Lifted Index
SLI	Surface lifted index
PrecipType	Surface Precipitation Type (SPT)
SFCR	Surface roughness
SSST	Surface Salinity Trend
SLTYP	Surface Slope Type
ModelSurfaceTemperature	Surface Temperature
SSTT	Surface Temperature Trend
SSTOR	Surface water storage
Surge	Surge Height
SX	Sweat index
TMPA	Temperature anomaly
T	Temperature
TMP	Temperature
TMPSWP	Temperature
RCT	Temperature parameter in canopy conductance
RCT	Temperature parameter in canopy conductance
TTDIA	Temperature Tendency By All Physics
TTRAD	Temperature tendency by all radiation
TTRAD	Temperature tendency by all radiation
TTPHY	Temperature Tendency By Non-radiation Physics
WTEND	Tendency of vertical velocity The Associated Legendre Functions of the first kind are defined
MASK	Thematic Mask
THICK	Thickness
TSC	Thunderstorm coverage
TSMT	Thunderstorm maximum tops
TSTM	Thunderstorm probability
TSTM	Thunderstorm Probability
TACONCP	Time-integrated air concentration of caesium pollutant
TACONIP	Time-integrated air concentration of iodine pollutant
TACONRDP	Time-integrated air concentration of radioactive pollutant
PTOR	Tornado probability
TORPROB	Tornado probability
TRNDPROB	Tornado probability
TCDC	Total cloud cover
TCOLI	Total column-integrated cloud ice
TCOLI	Total Column-Integrated Cloud Ice
TCOLW	Total column-integrated cloud water
TCOLW	Total Column-Integrated Cloud Water
TCOLC	Total column-integrated condensate
TCOLC	Total Column-Integrated Condensate
TCOLM	Total column-integrated melting ice
TCOLM	Total column-integrated melting ice
TCIOZ	Total Column Integrated Ozone
TCOLR	Total Column Integrated Rain
TCOLR	Total column integrated rain

Table 1 – continued from previo

Abbreviation	Description
TCOLR	Total Column Integrated Rain
TCOLS	Total Column Integrated Snow
TCOLS	Total column integrated snow
TCOLS	Total Column Integrated Snow
TCLSW	Total column-integrated supercooled liquid water
TCLSW	Total column-integrated supercooled liquid water
TCIWV	Total Column Integrated Water Vapour
TCIWV	Total Column Integrated Water Vapour
TCOLG	Total Column Integrate Graupel
TCWAT	Total Column Water (Vertically integrated total water (vapour+cloud water/ice)
TCWAT	Total Column Water(Vertically integrated total water (vapour+cloud water/ice)
TCOND	Total Condensate
TCOND	Total condensate
TCOND	Total Condensate
THFLX	Total Downward Heat Flux at Surface
TIPD	Total Icing Potential Diagnostic
TIPD	Total Icing Potential Diagnostic
TOZNE	Total ozone
A PCP	Total Precipitation
APCP	Total precipitation
APCPN	Total precipitation (nearest grid point)
APCPN	Total precipitation (nearest grid point)
TPRATE	Total Precipitation Rate
TPRATE	Total Precipitation Rate
PRSIGSV	Total Probability of Extreme Severe Thunderstorms (Days 2,3)
PRSIGSVR	Total Probability of Extreme Severe Thunderstorms (Days 2,3)
PRSVR	Total Probability of Severe Thunderstorms (Days 2,3)
ASNOW	Total Snowfall
TOTSN	Total snowfall
TSRATE	Total Snowfall Rate
TSRATE	Total Snowfall Rate
TSRWE	Total Snowfall Rate Water Equivalent
TSRWE	Total Snowfall Rate Water Equivalent
TSNOW	Total Snow
TSNOW	Total Snow
TSNOWP	Total Snow Precipitation
TSNOWP	Total Snow Precipitation
TTX	Total totals index
TWATP	Total Water Precipitation
TWATP	Total Water Precipitation
TTHDP	Transient thermocline depth
TRANSO	Transpiration Stree-Onset(Soil Moisture)
SMREF	Transpiration stress-onset (soil moisture)
TRANS	Transpiration
TCHP	Tropical Cyclone Heat Potential
TRO	Tropopause
TRBBS	Turbulence base
TURBB	Turbulence Base
TURB	Turbulence

Table 1 – continued from previo

Abbreviation	Description
TPFI	Turbulence Potential Forecast Index
TURB	Turbulence
TRBTP	Turbulence top
TURBT	Turbulence Top
TKE	Turbulent Kinetic Energy
TKE	Turbulent kinetic energy
UOGRD	u-component of current
UOGRD	u-component of current
MAXUW	U Component of Hourly Maximum 10m Wind Speed
UICE	u-component of ice drift
UGUST	u-component of wind gust
UGRD	u-component of wind
USTM	U-component storm motion
USTM	U-Component Storm Motion
USSD	U-component Surface Stokes Drift
UVI	Ultra Violet Index
UPHL	Updraft Helicity in Layer 2-5 km AGL
UPHL	Updraft Helicity
ULSM	Upper layer soil moisture
ULST	Upper layer soil temperature
ULWRF	Upward Long-Wave Rad. Flux
ULWRF	Upward long-wave radiation flux Upward Long-W/m^2
USWRF	Upward Short-Wave Rad. Flux
USWRF	Upward short-wave radiation flux Upward Short-W/m^2
UTRF	Upward Total radiation Flux
DUVB	UV-B downward solar flux
UVI	UV Index
UVIUCS	UV Index (Under Clear Sky)
VAPP	Vapor pressure
VAPP	Vapor Pressure
VOGRD	v-component of current
VOGRD	v-component of current
MAXVW	V Component of Hourly Maximum 10m Wind Speed
VICE	v-component of ice drift
UGUST	v-component of wind gust
VGRD	v-component of wind
VSTM	V-component storm motion
VSTM	V-Component Storm Motion
VSSD	V-component Surface Stokes Drift
VEGT	Vegetation canopy temperature
VGTYP	Vegetation Type
VEG	Vegetation
SPEED	Velocity Magnitude (Speed)
LMV	Velocity Point Model Surface
VPOT	Velocity potential
VRATE	Ventilation Rate
VDFHR	Vertical Diffusion Heating rate
VDFVA	Vertical Diffusion Meridional Acceleration
VDFMR	Vertical Diffusion Moistening Rate

Table 1 – continued from previo

Abbreviation	Description
VDFUA	Vertical Diffusion Zonal Acceleration
VEDH	Vertical Eddy Diffusivity Heat exchange
VTEC	Vertical Electron Content
VII	Vertically Integrated Ice (VII)
VILIQ	Vertically-integrated liquid
MRMSVILDensity	Vertically Integrated Liquid (VIL) Density
MRMSVIL	Vertically Integrated Liquid (VIL)
VIL	Vertically Integrated Liquid (VIL)
VWSH	Vertical speed shear
VWSH	Vertical speed sheer
VUCSH	Vertical u-component shear
VVCSH	Vertical v-component shear
DZDT	Vertical velocity geometric
VVEL	Vertical velocity pressure
VPTMP	Virtual potential temperature
VTMP	Virtual temperature
VIS	Visibility
VBDSF	Visible Beam Downward Solar Flux
SBSALB	Visible, Black Sky Albedo
VDDSF	Visible Diffuse Downward Solar Flux
Visible	Visible Imagery
SWSALB	Visible, White Sky Albedo
VASH	Volcanic ash
VAFTD	Volcanic Ash Forecast Transport and Dispersion
VOLASH	Volcanic Ash
VOLDEC	Volumetric Direct Evaporation Cease(Soil Moisture)
VSOSM	Volumetric Saturation Of Soil Moisture
SOILW	Volumetric soil moisture content
VSOILM	Volumetric Soil Moisture
VOLTSO	Volumetric Transpiration Stree-Onset(Soil Moisture)
VWILTM	Volumetric Wilting Moisture
WCINC	Water condensate added by precip assimilation
WCCONV	Water Condensate Flux Convergance (Vertical Int)
WCVFLX	Water Condensate Meridional Flux (Vertical Int)
WCUFLX	Water Condensate Zonal Flux (Vertical Int)
WEASD	Water Equivalent of Accumulated Snow Depth
WEASD	Water equivalent of accumulated snow depth
WATR	Water runoff
TEMPWTR	Water temperature
WVINC	Water vapor added by precip assimilation
WVCONV	Water Vapor Flux Convergance (Vertical Int)
WaterVapor	Water Vapor Imagery
WVVFLX	Water Vapor Meridional Flux (Vertical Int)
WVUFLX	Water Vapor Zonal Flux (Vertical Int)
WDIRW	Wave Directional Width
WESP	Wave Engery Spectrum
WVSP1	Wave spectra (1)
WVSP2	Wave spectra (2)
WVSP3	Wave spectra (3)

Table 1 – continued from previo

Abbreviation	Description
WSTP	Wave Steepness
WSTR	Wave Stress
wxType	Weather
ModelWetbulbTemperature	Wet Bulb Temperature
WHTCOR	White Light Coronagraph Radiance
WHTRAD	White Light Radiance
WILT	Wilting Point
WILT	Wilting point
WCI	Wind chill factor
WDIR	Wind direction (from which blowing)
WMIXE	Wind mixing energy
WINDPROB	Wind probability
WINDPROB	Wind Probability
WGS	Wind speed gust
PWS	Wind speed
WIND	Wind speed
HGT X	X-gradient of Height
LPS X	X-gradient of Log Pressure
XRAYRAD	X-Ray Radiance
HGT Y	Y-gradient of Height
LPS Y	Y-gradient of Log Pressure
UGWD	Zonal flux of gravity wave stress
U-GWD	Zonal Flux of Gravity Wave Stress

About Unidata AWIPS

AWIPS is a weather forecasting display and analysis package developed by the National Weather Service and Raytheon. AWIPS is a Java application consisting of a data-rendering client (CAVE, which runs on Red Hat/CentOS Linux and Mac OS X) and a backend data server (EDEX, which runs only on Linux)

AWIPS takes a unified approach to data ingest, and most data types follow a standard path through the system. At a high level, data flow describes the path taken by a piece of data from its source to its display by a client system. This path starts with data requested and stored by an *LDM* client and includes the decoding of the data and storing of decoded data in a form readable and displayable by the end user.

The AWIPS ingest and request processes are a highly distributed system, and the messaging broken *Qpid* is used for inter-process communication.

License

About AWIPS

The primary AWIPS application for data ingest, processing, and storage is the Environmental Data EXchange (**EDEX**) server; the primary AWIPS application for visualization/data manipulation is the Common AWIPS Visualization En-

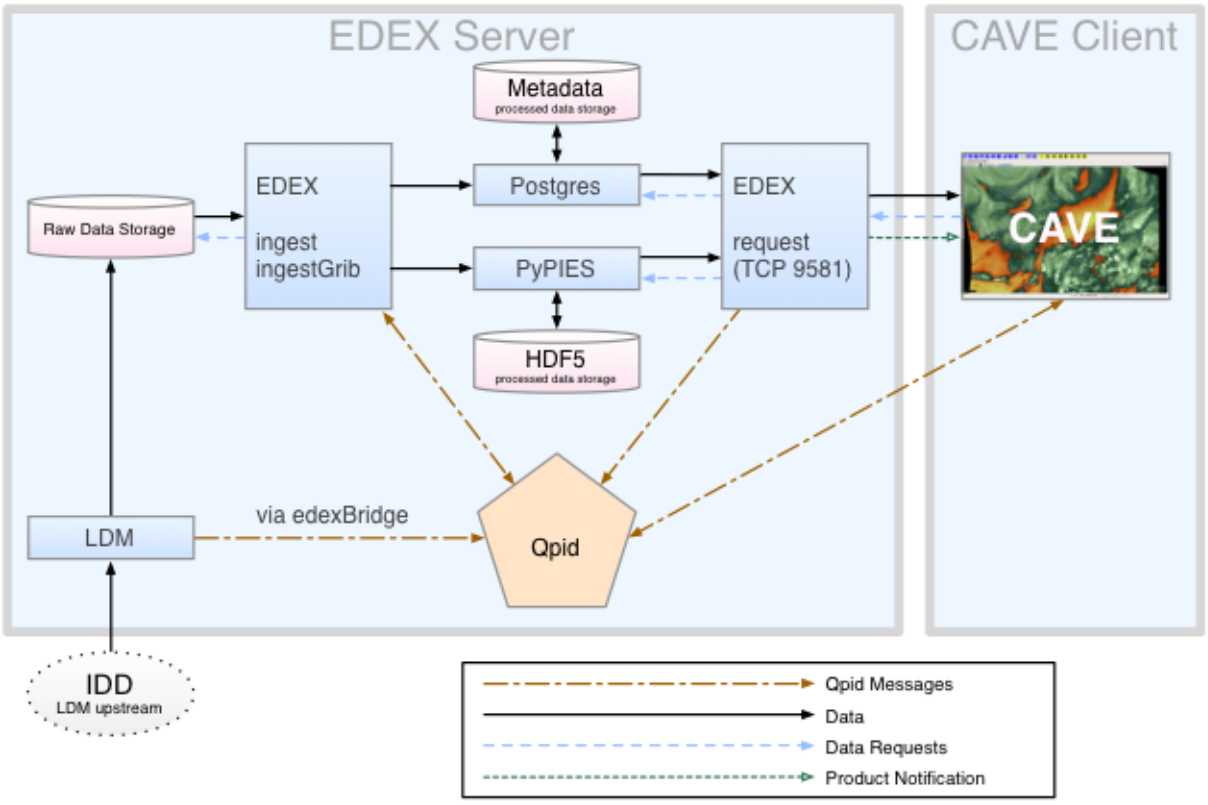

EDEX Server
CAVE Client
Metadata
processed data storage
EDEX
ingest
ingestGrib
Postgres
EDEX
request
(TCP 9581)
Raw Data Storage
PyPIES
CAVE
HDF5
processed data storage
via edexBridge
LDM
Qpid
IDD
LDM upstream
Qpid Messages
Data
Data Requests
Product Notification

Fig. 1: image

vironment (**CAVE**) client, which is typically installed on a workstation separate from other AWIPS components.

In addition to programs developed specifically for AWIPS, AWIPS uses several commercial off-the-shelf (COTS) and Free or Open Source software (FOSS) products to assist in its operation. The following components, working together and communicating, compose the entire AWIPS system.

EDEX

The main server for AWIPS. Qpid sends alerts to EDEX when data stored by the LDM is ready for processing. These Qpid messages include file header information which allows EDEX to determine the appropriate data decoder to use. The default ingest server (simply named ingest) handles all data ingest other than grib messages, which are processed by a separate ingestGrib server. After decoding, EDEX writes metadata to the database via Postgres and saves the processed data in HDF5 via PyPIES. A third EDEX server, request, feeds requested data to CAVE clients. EDEX ingest and request servers are started and stopped with the commands `edex start` and `edex stop`, which runs the system script `/etc/rc.d/init.d/edex_camel`

CAVE

Common AWIPS Visualization Environment. The data rendering and visualization tool for AWIPS. CAVE contains of a number of different data display configurations called perspectives. Perspectives used in operational forecasting environments include **D2D** (Display Two-Dimensional), **GFE** (Graphical Forecast Editor), and **NCP** (National Centers Perspective). CAVE is started with the command `/awips2/cave/cave.sh` or `cave.sh`

Alertviz

Alertviz is a modernized version of an AWIPS I application, designed to present various notifications, error messages, and alarms to the user (forecaster). AlertViz can be executed either independently or from CAVE itself. In the Unidata CAVE client, Alertviz is run within CAVE and is not required to be run separately. The toolbar is also **hidden from view** and is accessed by right-click on the desktop taskbar icon.

LDM

http://www.unidata.ucar.edu/software/ldm/

The **LDM** (Local Data Manager), developed and supported by Unidata, is a suite of client and server programs designed for data distribution, and is the fundamental component comprising the Unidata Internet Data Distribution (IDD) system. In AWIPS, the LDM provides data feeds for grids, surface observations, upper-air profiles, satellite and radar imagery and various other meteorological datasets. The LDM writes data directly to file and alerts EDEX via Qpid when a file is available for processing. The LDM is started and stopped with the commands `edex start` and `edex stop`, which runs the commands `service edex_ldm start` and `service edex_ldm stop`

edexBridge

edexBridge, invoked in the LDM configuration file `/awips2/ldm/etc/ldmd.conf`, is used by the LDM to post "data available" messaged to Qpid, which alerts the EDEX Ingest server that a file is ready for processing.

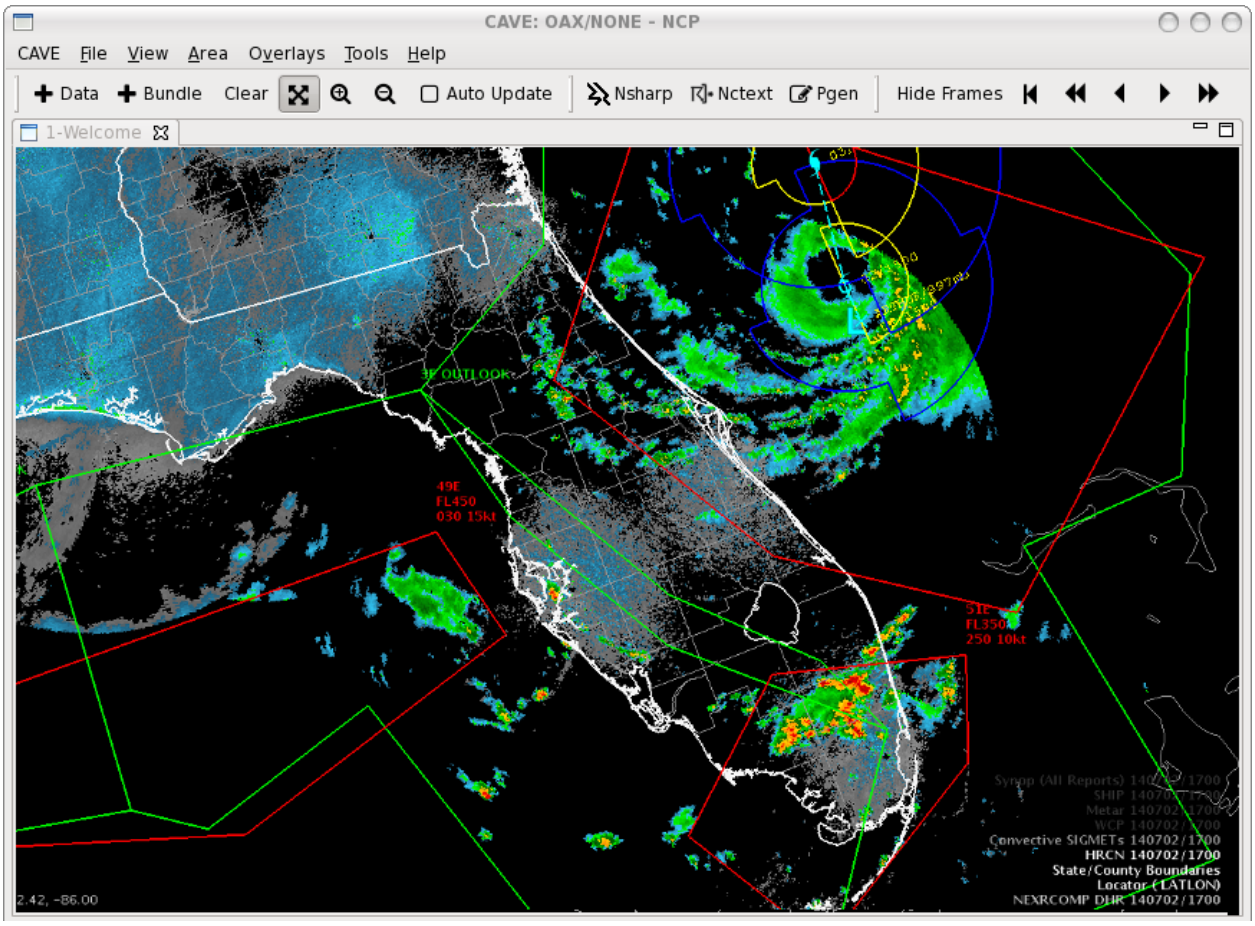

CAVE: OAX/NONE - NCP
CAVE File View Area Overlays Tools Help
Data Bundle Clear Auto Update Nsharp Nctext Pgen Hide Frames
1-Welcome
49E
FL450
030 15kt
51E
FL350
250 10kt
Convective SIGMETs 140702/1700
HRCN 140702/1700
State/County Boundaries
Locator (LATLON)
NEXRCOMP DHR 140702/1700
2.42, -86.00

Fig. 2: CAVE

Qpid

http://qpid.apache.org

Apache Qpid, the Queue Processor Interface Daemon, is the messaging system used by AWIPS to facilitate communication between services. When the LDM receives a data file to be processed, it employs **edexBridge** to send EDEX ingest servers a message via Qpid. When EDEX has finished decoding the file, it sends CAVE a message via Qpid that data are available for display or further processing. Qpid is started and stopped by `edex start` and `edex stop`, and is controlled by the system script `/etc/rc.d/init.d/qpidd`

PostgreSQL

http://www.postgresql.org

PostgreSQL, known simply as Postgres, is a relational database management system (DBMS) which handles the storage and retrieval of metadata, database tables and some decoded data. The storage and reading of EDEX metadata is handled by the Postgres DBMS. Users may query the metadata tables by using the termainal-based front-end for Postgres called **psql**. Postgres is started and stopped by `edex start` and `edex stop`, and is controlled by the system script `/etc/rc.d/init.d/edex_postgres`

HDF5

http://www.hdfgroup.org/HDF5/

Hierarchical Data Format (v.5) is the primary data storage format used by AWIPS for processed grids, satellite and radar imagery and other products. Similar to netCDF, developed and supported by Unidata, HDF5 supports multiple types of data within a single file. For example, a single HDF5 file of radar data may contain multiple volume scans of base reflectivity and base velocity as well as derived products such as composite reflectivity. The file may also contain data from multiple radars. HDF5 is stored in `/awips2/edex/data/hdf5/`

PyPIES (httpd-pypies)

PyPIES, Python Process Isolated Enhanced Storage, was created for AWIPS to isolate the management of HDF5 Processed Data Storage from the EDEX processes. PyPIES manages access, i.e., reads and writes, of data in the HDF5 files. In a sense, PyPIES provides functionality similar to a DBMS (i.e PostgreSQL for metadata); all data being written to an HDF5 file is sent to PyPIES, and requests for data stored in HDF5 are processed by PyPIES.

PyPIES is implemented in two parts: 1. The PyPIES manager is a Python application that runs as part of an Apache HTTP server, and handles requests to store and retrieve data. 2. The PyPIES logger is a Python process that coordinates logging. PyPIES is started and stopped by `edex start` and `edex stop`, and is controlled by the system script `/etc/rc.d/init.d/https-pypies`

Python Module Index

a

Index

I

L

M

N

P

S

T