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
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
Requirements

• python 2.7+
• numpy
• shapely
• six

3.1 Quick Example

```python
from awips.dataaccess import DataAccessLayer
DataAccessLayer.changeEDEXHost("edex-cloud.unidata.ucar.edu")
dataTypes = DataAccessLayer.getSupportedDatatypes()
list(dataTypes)
['acars',
 'binlightning',
 'bufrmosavn',
 'bufrmoseta',
 'bufrmosgfs',
 'bufrmoshpc',
 'bufrmoslamp',
 'bufrmosmrfs',
 'bufrua',
 'climate',
 'common_obs_spatial',
 'gfe',
 'gfeeditarea',
 'grid',
 'maps',
 'modelsounding',
 'obs',
 'practicewarning',
]```

(continues on next 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 identifiers for NEXRAD3 products.

Args: availableParms: Full list of radar parameters
Returns: List of filtered parameters

awips.dataaccess.DataAccessLayer.getRadarProductNames(availableParms)
Get only the named identifiers 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

3.1. Quick Example
**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)*

**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)*

**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 DateTime, then by level and finally by weather element. If no data is available for the given request parameters, None is returned.

3.1. Quick Example
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.dataaccessCombinedTimeQuery.getAvailableTimes(request, refTimeOnly=False)

• genindex
Available Data Types

satellite

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

```python
# Construct 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:

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

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

```python
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:
import DataAccessLayer

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)
fctRun = DataAccessLayer.getForecastRun(cycles[-1], times)
response = DataAccessLayer.getGridData(request, [fctRun[-1]])

for grid in response:
    data = grid.getRawData()
    lons, lats = grid.getLatLonCoords()

---

**warning**

- Shapely MultiPolygon, Polygon:

```python
MULTIPOLYON ((-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:

```python
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:

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

```python
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() + " "
```
with `contourf`

```python
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

```python
%matplotlib inline

Notebook

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

```python
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)
```python
(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():
    ...
```

3.1. Quick Example
```python
# Extract temperature values and replace -9999.0 with 'nan'
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

![Map of temperature data](image.png)
## 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
  
  ```python
  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.

```python
%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)
sndObject = ModelSounding.getSounding(model, useParms, "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)) + "]")

# 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)
# 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")
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)
```python
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

3.1. Quick Example
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
- GLMf1 Flashes
- GLMgr Groups

and with seven attributes:

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

GLM Sources and Parameters

```python
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 = DataAccessAccessLayer.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.
```python
from awips.dataaccess import DataAccessLayer
import unittest

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

```

DataAccessLayer.getAvailableLocationNames()

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

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

```

(continues on next 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_HAWAIII_1KM',
'HFR-US_HAWAIII_2KM',
'HFR-US_HAWAIII_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()`

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


(continues on next page)
'EHIi',
'EPT',
'EPTA',
'EPTC',
'EPTGrd',
'EPTGrdM',
'EPTs',
'EPVg',
'EPVs',
'EPVt1',
'EPVt2',
'ESP',
'ESP2',
'FRZR12hr',
'FRZRrun',
'FVecs',
'FeatMot',
'Fnvecs',
'FsVecs',
'Fzra1',
'Fzra2',
'GH',
'GHexSM',
'GHexSM2',
'Gust',
'HI',
'HII',
'H13',
'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)
'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',

3.1. Quick Example
'SNW',
'SNWA',
'SRM1',
'SRM1M',
'SRMm',
'SRMmM',
'SRMn',
'SRMnM',
'SSP',
'SSi',
'STP',
'STP1',
'SShear',
'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)
'TotQi',
'Tstk',
'TwMax',
'TwMin',
'Twstb',
'TxSM',
'USTM',
'VAdv',
'VAdvAdvection',
'VG3',
'VSTM',
'Vis',
'WCD',
'WD',
'WEASD',
'WEASD1hr',
'WGS',
'Wind',
'WndChl',
'ageoVC',
'ageoW',
'ageoWM',
'cCape',
'cCln',
'cTOT',
'capeToLvl',
'dCape',
'dGHI2',
'dP',
'dP1hr',
'dP3hr',
'dP6hr',
'dPW1hr',
'dPW3hr',
'dPW6hr',
'dT',
'dVAdv',
'dZ',
'defV',
'del12gH',
'df',
'fGen',
'fnD',
'fsD',
'gamma',
'gammaE',
'geoVort',
'geoW',
'geoWM',
'locape',
'maxEPT',
'minEPT',
'mixRat',
'msl-P',
'muCape',
'pV',
'pVeq',

(continues on next page)
'qDiv',
'qVec,'
'qmVec',
'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)
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)
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0_30.0BL</td>
<td>310.0_320.0Ke</td>
</tr>
<tr>
<td>310.0Ke</td>
<td>330.0K</td>
</tr>
<tr>
<td>900.0_800.0MB</td>
<td>550.0_500.0MB</td>
</tr>
<tr>
<td>2.4TILT</td>
<td>50.0TILT</td>
</tr>
<tr>
<td>3500.0FHAG</td>
<td>35.0TILT</td>
</tr>
<tr>
<td>12.0TILT</td>
<td>300.0_310.0K</td>
</tr>
<tr>
<td>3000.0_12000.0FHAG</td>
<td>0.9TILT</td>
</tr>
<tr>
<td>320.0K</td>
<td>400.0_350.0MB</td>
</tr>
<tr>
<td>500.0FHAG</td>
<td>750.0_700.0MB</td>
</tr>
<tr>
<td>1000.0_400.0MB</td>
<td>345.0K</td>
</tr>
<tr>
<td>250.0_260.0K</td>
<td>300.0Ke</td>
</tr>
<tr>
<td>290.0Ke</td>
<td>950.0_900.0MB</td>
</tr>
<tr>
<td>4572.0FHAG</td>
<td>275.0_285.0Ke</td>
</tr>
<tr>
<td>335.0Ke</td>
<td>295.0_305.0Ke</td>
</tr>
<tr>
<td>275.0_285.0K</td>
<td>600.0_550.0MB</td>
</tr>
<tr>
<td>310.0K</td>
<td>9000.0FHAG</td>
</tr>
<tr>
<td>335.0K</td>
<td>1000.0_7000.0FHAG</td>
</tr>
<tr>
<td>700.0_500.0MB</td>
<td>9144.0FHAG</td>
</tr>
<tr>
<td>325.0_335.0K</td>
<td>300.0_2000.0FHAG</td>
</tr>
<tr>
<td>2200.0_8000.0FHAG</td>
<td>0.0_1000.0FHAG</td>
</tr>
<tr>
<td>0.0_609.6FHAG</td>
<td>0.0MAXOMEGA</td>
</tr>
<tr>
<td>315.0_325.0K</td>
<td>325.0K</td>
</tr>
<tr>
<td>340.0Ke</td>
<td>0.0_4000.0FHAG</td>
</tr>
<tr>
<td>5000.0_5500.0FHAG</td>
<td>300.0_250.0MB</td>
</tr>
<tr>
<td>1.5TILT</td>
<td>335.0_345.0K</td>
</tr>
<tr>
<td>2.0FHAG</td>
<td>315.0K</td>
</tr>
<tr>
<td>3.4TILT</td>
<td>2500.0FHAG</td>
</tr>
<tr>
<td>10000.0FHAG</td>
<td>0.0_2000.0FHAG</td>
</tr>
<tr>
<td>7000.0FHAG</td>
<td>0.0_1000.0FHAG</td>
</tr>
</tbody>
</table>
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)
285.0 Ke
290.0 K
305.0_315.0 K
285.0_295.0 K
0.0_2500.0 FHAG
925.0_850.0 MB
275.0 Ke
1500.0_2000.0 FHAG
2.0 PV
300.0_200.0 MB
610.0_40000.0 FHAG
260.0_270.0 K
0.0_6000.0 FHAG
2743.2 FHAG
3000.0 FHAG
315.0_325.0 Ke
600.0_500.0 MB
16.7 TILT
0.5 PV
280.0 K
500.0_250.0 MB
40.0 TILT
1000.0 FHAG
3048.0 FHAG
400.0_200.0 MB
300.0_310.0 Ke
270.0_280.0 K
1000.0_700.0 MB
45.0 TILT
850.0_500.0 MB
60.0_90.0 BL
2500.0_3000.0 FHAG
609.6 FHAG
0.0_8000.0 FHAG
295.0 K
4.3 TILT
295.0_305.0 K
330.0_340.0 Ke
270.0 K
4000.0_4500.0 FHAG
280.0_290.0 K
925.0_700.0 MB
0.0_1500.0 FHAG
260.0 K
10.0 TILT
3500.0_4000.0 FHAG
325.0 Ke
285.0 K
290.0_300.0 Ke
7.5 TILT
1828.8 FHAG
280.0 Ke
500.0_450.0 MB
305.0_315.0 Ke
250.0 K
4500.0 FHAG
1250.0 FHAG

(continues on next page)
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 (F:ff)
- **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 DateTime array for a single forecast cycle.

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

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

```python
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

```python
%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
```
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()))

3.1. Quick Example
This exercise creates a METAR plot for North America using AWIPS METAR observations (datatype `obs`) and MetPy.

```python
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)
```
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', 'OYL', 'SYR', 'LEX', 'CHS', 'TLH', 'HOU', 'GJT', 'LBB', 'LSV',
           'GRB', 'CLT', 'LNK', 'DSM', 'BOI', 'FSD', 'RAP', 'RIC', 'JAN', 'HSV', 'CRW',
           'SAT', 'BUY', 'OCA', 'ZPC', 'VIR', 'BGD', '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', 'FLL', 'CLE', 'UNV']
selected = [K.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):

```python
# 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)
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.getNumber("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 == '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

```python
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'
```
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'])

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

```python
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.

```python
# 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)
# 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

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')
avx.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='-', edgcolor='black', linewidth=2)
av.add_feature(shape_feature)

Using 23 county MultiPolygons
Create a merged CWA with cascaded_union

```python
# 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)
```

3.1. Quick Example
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.

```python
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.

```python
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)
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

```python
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

```python
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)
```

Chapter 3. Requirements
Topography

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

```python
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:
" + str(gridData[0].getRawData().shape) + 
"Sample grid data:
" + str(gridData[0].getRawData()) + "
)"
```
Number of grid records: 1
Sample grid data shape: (778, 1058)

Sample grid data:

$$\begin{bmatrix} 1694. & 1693. & 1688. & \ldots & 757. & 761. & 762. \\
1701. & 1701. & 1701. & \ldots & 758. & 760. & 762. \\
1703. & 1703. & 1703. & \ldots & 760. & 761. & 762. \\
\vdots \\
1767. & 1741. & 1706. & \ldots & 769. & 762. & 768. \\
1767. & 1746. & 1716. & \ldots & 775. & 765. & 761. \\
1781. & 1753. & 1730. & \ldots & 766. & 762. & 759. \end{bmatrix}$$

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

```python
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)
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.plot 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)
'AQQ',
'AQH',
'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',
'CD1',

(continues on next 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',
'CLE',
'CLN',
'COL1',
'COL2',
'COL3',
'COL4',
'COT',
'CQV',
'CRL',
'CRR',
'CTY',
'CVM',
'CVS',
'CWEU',
'CWFN',
'CWKH',
'CWLB',
'CWLO',
'CWLT',
(continues on next page)
<table>
<thead>
<tr>
<th>CWLW</th>
<th>CWMW</th>
<th>CWOS</th>
<th>CWPB</th>
<th>CWQG</th>
<th>CWSA</th>
<th>CWSE</th>
<th>CWZB</th>
<th>CWZC</th>
<th>CWZV</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYAH</td>
<td>CYAW</td>
<td>CYBK</td>
<td>CYBU</td>
<td>CYCB</td>
<td>CYCC</td>
<td>CYCX</td>
<td>CYDA</td>
<td>CYEG</td>
<td>CYEV</td>
</tr>
<tr>
<td>CYFB</td>
<td>CYFO</td>
<td>CYFS</td>
<td>CYGQ</td>
<td>CYHM</td>
<td>CYHZ</td>
<td>CYJT</td>
<td>CYLH</td>
<td>CYLJ</td>
<td>CYMD</td>
</tr>
<tr>
<td>CYMO</td>
<td>CYMT</td>
<td>CYMX</td>
<td>CYOC</td>
<td>CYOW</td>
<td>CYPA</td>
<td>CYPE</td>
<td>CYPL</td>
<td>CYPO</td>
<td>CYQA</td>
</tr>
<tr>
<td>CYQD</td>
<td>CYQG</td>
<td>CYQH</td>
<td>CYQI</td>
<td>CYQK</td>
<td>CYQQ</td>
<td>CYQR</td>
<td>CYQT</td>
<td>CYQX</td>
<td>CYQY</td>
</tr>
<tr>
<td>CYRB</td>
<td>CYSM</td>
<td>CYTH</td>
<td>CYTL</td>
<td>CYTS</td>
<td>CYUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
'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',
'CYZU',
'DEN',
'DOV',
'DPG',
'DSC',
'DSD',
'DTX',
'DVN',
'DYS',
'E28',
'E74',
'EAR',
'EAX',
'EDW',
'EFL',
'EMP',
'END',
'ENL',
'ESTC',
'FCS',
'FDR',
'FDC',
'FHI',
'FLG',
'FLP',
'FPK',
'FRI',
'FSI',
'FTR',

(continues on next page)
3.1. Quick Example
'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',
'KAW',
'KAVL',
'KAVP',
'KAXN',
'KAYS',
'KAZO',
'KBAF',
'KBCE',
'KBDE',
'KBDL',
'KBDR',
'KBED',
'KFBD',
'KFFF',
'KFI',
'KFIL',
'KBGM',
'KBGR',
'KBHB',
'KBHM',
'KBIH',
'KBIL',
'KBIS',
'KBJC',
'KBJI',
'KBKE',
(continues on next page)
'KBKW',
'KBLF',
'KBLH',
'KBLI',
'KBML',
'KBNA',
'KBNO',
'KBNV',
'KBOI',
'KBOS',
'KBPT',
'KBQK',
'KBRD',
'KBRL',
'KBRO',
'KBTL',
'KBTM',
'KBTM',
'KBTV',
'KBVF',
'KBUR',
'KBVI',
'KBVX',
'KBVY',
'KBMW',
'KBWI',
'KBXY',
'KBZN',
'KCAE',
'KCAK',
'KCAR',
'KCDC',
'KCDR',
'KCDT',
'KCEC',
'KCEF',
'KCGI',
'KCGX',
'RCHA',
'RCHH',
'RCHO',
'RCHS',
'KCID',
'KCIU',
'KCKB',
'KCKL',
'KCLE',
'KCLI',
'KCLM',
'KCLT',
'KCMH',
'KCMJ',
'KCMX',
'KCNM',
'KCNU',
'KCOD',
'KCOE',

(continues on next page)
'KCON',
'KCOS',
'KCOU',
'KCRP',
'KCRE',
'KCRQ',
'KCRW',
'KCSG',
'KCSV',
'KCTB',
'KCVG',
'KCWA',
'KCYS',
'KDAB',
'KDAG',
'KDAL',
'KDAN',
'KDAY',
'KDBQ',
'KDCA',
'KDDC',
'KDEC',
'KDEN',
'KDET',
'KDFW',
'RDHN',
'RDHT',
'RDII',
'RDID',
'RDLM',
'RDLS',
'RDMM',
'RDPA',
'RDRA',
'RDRO',
'RDSM',
'RDTW',
'RDFU',
'RDUJ',
'REAT',
'REAU',
'RECG',
'REED',
'REGE',
'REKN',
'REKO',
'REL',
'RELD',
'RELM',
'RELO',
'RELP',
'RELY',
'RENV',
'REPH',
'REPO',
'REPZ',
'REQ


<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>KERI</td>
</tr>
<tr>
<td>KESF</td>
</tr>
<tr>
<td>KEUG</td>
</tr>
<tr>
<td>KEVV</td>
</tr>
<tr>
<td>KEWB</td>
</tr>
<tr>
<td>KEWN</td>
</tr>
<tr>
<td>KEWR</td>
</tr>
<tr>
<td>KEYW</td>
</tr>
<tr>
<td>KFAM</td>
</tr>
<tr>
<td>KFAR</td>
</tr>
<tr>
<td>KFAT</td>
</tr>
<tr>
<td>KFAY</td>
</tr>
<tr>
<td>KFCA</td>
</tr>
<tr>
<td>KFDY</td>
</tr>
<tr>
<td>KFKL</td>
</tr>
<tr>
<td>KFLG</td>
</tr>
<tr>
<td>KFLL</td>
</tr>
<tr>
<td>KFLO</td>
</tr>
<tr>
<td>KFMN</td>
</tr>
<tr>
<td>KFMY</td>
</tr>
<tr>
<td>KFNT</td>
</tr>
<tr>
<td>KFGE</td>
</tr>
<tr>
<td>KFPR</td>
</tr>
<tr>
<td>KFRM</td>
</tr>
<tr>
<td>KFSM</td>
</tr>
<tr>
<td>KFTW</td>
</tr>
<tr>
<td>KFTY</td>
</tr>
<tr>
<td>KFVE</td>
</tr>
<tr>
<td>KFVX</td>
</tr>
<tr>
<td>KFWA</td>
</tr>
<tr>
<td>KFXE</td>
</tr>
<tr>
<td>KFYV</td>
</tr>
<tr>
<td>KGAG</td>
</tr>
<tr>
<td>KGCC</td>
</tr>
<tr>
<td>KGCK</td>
</tr>
<tr>
<td>KGCK</td>
</tr>
<tr>
<td>KGCN</td>
</tr>
<tr>
<td>KGEG</td>
</tr>
<tr>
<td>KGFK</td>
</tr>
<tr>
<td>KGFL</td>
</tr>
<tr>
<td>KGGL</td>
</tr>
<tr>
<td>KGGM</td>
</tr>
<tr>
<td>KGGT</td>
</tr>
<tr>
<td>KGJT</td>
</tr>
<tr>
<td>KGLD</td>
</tr>
<tr>
<td>KGLH</td>
</tr>
<tr>
<td>KGLS</td>
</tr>
<tr>
<td>KGML</td>
</tr>
<tr>
<td>KGMR</td>
</tr>
<tr>
<td>KGNT</td>
</tr>
<tr>
<td>KGNO</td>
</tr>
<tr>
<td>KGPT</td>
</tr>
<tr>
<td>KGPB</td>
</tr>
<tr>
<td>KGRI</td>
</tr>
<tr>
<td>KGRR</td>
</tr>
<tr>
<td>KGSO</td>
</tr>
<tr>
<td>KGSP</td>
</tr>
<tr>
<td>KGTF</td>
</tr>
</tbody>
</table>

(continues on next 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',
'KL2K',
'KMAF',
'RMBS',
'RMCB',
'RMCE',
'RMCI',
'KMCN',
'KMCO',
'KMCW',
'KMDN',
'KMDT',
'KMDW',
'KMEI',
'KMEM',
'KMFD',
'KMFE',
'KMFR',
'KMGM',
'KMGW',
'KMHE',
'KMHK',

(continues on next page)
"KMHT",
"KMHX",
"KMIA",
"KMIV",
"KMKC",
"KMKE",
"KMKG",
"KMKL",
"KMLB",
"KMLC",
"KMLI",
"KMLS",
"KMLT",
"KMLU",
"KMMU",
"KMOB",
"KMOT",
"KMFV",
"KMQT",
"KMRB",
"KMRY",
"KMSL",
"KMSN",
"KMSO",
"KMSF",
"KMSI",
"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",
"KPI",
"KPK",
"KPDK",

(continued from previous page)
'KPDT',
'KPDX',
'KPFN',
'KPGA',
'KPHF',
'KPHL',
'KPHN',
'KPHX',
'KPIA',
'KPIB',
'KPIE',
'KPIH',
'KPIR',
'KPIU',
'KPKB',
'KPLN',
'KPMD',
'KPNC',
'KPNE',
'KPNS',
'KPOU',
'KPOI',
'KPRB',
'KPRC',
'KPSM',
'KPSM',
'KPSP',
'KPTK',
'KPUU',
'KPVJ',
'KPWD',
'KPVI',
'KPWM',
'KRAD',
'KRAP',
'KRBL',
'KRDD',
'KRDG',
'KRDM',
'KRDU',
'KRFD',
'KRIC',
'KRIF',
'KRRM',
'KRRK',
'KRSK',
'KRNO',
'KRNT',
'KRGA',
'KRGC',
'KROW',
'KRSK',
'KSSL',
'KRSI',
'KRSW',
'KRUM',
'KRWF',
'KRWI',
'KWNL',
'KSWC',

3.1. Quick Example 79
'KSAF',
'KSAN',
'KSAT',
'KSAV',
'KSBA',
'KSBN',
'KSBP',
'KSBY',
'KSCH',
'KSCK',
'KSDF',
'KSDM',
'KSDY',
'KSEA',
'KSEP',
'KSFF',
'KSFO',
'KSGF',
'KSFU',
'KSHR',
'KSHV',
'KSJU',
'KSJT',
'KSLC',
'KSLM',
'KSLV',
'KSLK',
'KSLN',
'KSMF',
'KSMX',
'KSN',
'KSN',
'KSPI',
'KSPS',
'KSRQ',
'KSSI',
'KSTJ',
'KSTL',
'KSTP',
'KSTS',
'KSW',
'KSWX',
'KSY',
'KSYR',
'KTCC',
'KTCL',
'KTCS',
'KTEB',
'KTIW',
'KLHL',
'KTM',
'KTOL',
'KTOP',
'KTPA',
'KTPA',
'KTR',
'KTRI',
'KTRK',
'KTFRM',
'KTTFD',
'KTTTD',
'KTTTN',
'KTTUL',
'KTUPT',
'KTUS',
'KTV2C',
'KTVL',
'KTVW',
'KTXK',
'KTYR',
'KTVS',
'KUCA',
'KUIN',
'KUKI',
'KUNV',
'KVCT',
'KVEL',
'KVLD',
'KVNY',
'KVRB',
'KWJF',
'KWMC',
'KWR1',
'R2W',
'R26',
'R2K',
'R2N',
'R2N',
'KYNG',
'KYNM',
'KZ2V',
'LAA',
'LAP',
'LBY',
'LDL',
'LHX',
'LIC',
'LOJ',
'LRR',
'LSF',
'LUS',
'LVM',
'LWI',
'MAC',
'MAX',
'MAZ',
'MDPC',
'MDPP',
'MDSD',
'MDST',
'MGFL',
'MGHT',
'MGHT',
'MGPB',
'MGJS',

(continues on next page)
'MHAM',
'MHCA',
'MHCH',
'MHLC',
'MHLE',
'MHLM',
'MHNJ',
'MHPL',
'MHRO',
'MHSR',
'MHTE',
'MHTG',
'MHYR',
'MIB',
'MIE',
'MKJP',
'MKJS',
'MLD',
'MMAA',
'MMAS',
'MMBT',
'MMCE',
'MMCL',
'MMCM',
'MMCS',
'MMCU',
'MMCV',
'MMCZ',
'MMDO',
'MMGL',
'MMGM',
'MMHO',
'MMLP',
'MMMA',
'MMMD',
'MMML',
'MMMR',
'MMT',
'MMXX',
'MMY',
'MMZZ',
'MNNL',
'MMPR',
'MMRX',
'MMSD',
'MMSP',
'MMTC',
'MMTJ',
'MMTM',
'MMTO',
'MMTP',
'MMUN',
'MMVR',
'MMZC',
'MMZH',
'MMZO',
'MMGG',

(continues on next page)
]
```python
def _get_forecast_run(request, cycles, times):
    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, moisture (spec. humidity), wind components, and cloud cover)

```python
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'))
```

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

```python
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
\begin{verbatim}
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


# 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
\end{verbatim}

**MetPy SkewT and Hodograph**

```python
%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_barbs(p, u, v)
skew.ax.set_xlim(-40, 60)
skew.ax.set_ylim(1000, 100)

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

(continues on next 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

```python
import warnings
```

(continues on next page)
from awips.dataaccess import DataAccessLayer
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import numpy as np

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

dict

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)
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()
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 Refl1
Range: -27.5 to 51.5 (Unit : dBZ)
Size : (230, 360)
3.1. Quick Example
Chapter 3. Requirements
3.1. Quick Example
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)
3.1. Quick Example

---

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)
Chapter 3. Requirements
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)

3.1. Quick Example
Chapter 3. Requirements
3.1. Quick Example
Chapter 3. Requirements
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)

3.1. Quick Example
python-awips Documentation

Chapter 3. Requirements
### 3.1. Quick Example

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.

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

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)

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)
Determine lat/lon of maximum rainfall value:

```python
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

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

```python
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)
fcsRun = DataAccessLayer.getForecastRun(cycles[-2], times)

for i, tt in enumerate(fcsRun):
    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()]
fcsHr = int(grid.getDataTime().getFcstTime()/3600)

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

(continues on next page)
```python
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()))
```

```
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

```python
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
import metpy.units as 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)
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

```python
%matplotlib inline

This exercise creates a surface observation 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.

```
```python
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
```

```python
# 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()
```

```python
# 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")
```

```python
# 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)
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.

```python
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']

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.

```python
# Create envelope geometry
tenvelope = 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)
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

```python
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'),
                      np.array(obs['windDir'])* 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)
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)
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 == -9999.0] = '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)
am_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='0.1f', units='degF', color='darkred')
custom_layout.add_value('SW', 'dew_point_temperature', fmt='0.1f', 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)
Plot both METAR and SYNOP

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

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

```python
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)
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)
availableProducts.sort()
for product in availableProducts:
    print(" - " + product)
- 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-535.92hPa
- VMP-555.17hPa
- VMP-56.13hPa
- VMP-575.52hPa
- VMP-596.31hPa
- VMP-6.96hPa

(continues on next page)
3.1. Quick Example

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
- 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)
- 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)
- 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
- 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
  - Grid Cloud Amount
  - Grid 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)
- 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
GOES 16 Mesoscale Sectors

Define our imports, and define our map properties first.

```python
%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"
```

3.1. Quick Example

(continues on next 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)
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>
3.1. Quick Example
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.

```python
%matplotlib inline
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
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", "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([])

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

```python
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:

```python
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

```python
%matplotlib inline
bbox=[-127,-64,24,49]
fig, ax = make_map(bbox=bbox)
siteids=np.array([])
periods=np.array([])
```

(continues on next 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))

        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')
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:

```java
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:

```java
public DateTime[] 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:

```xml
<bean id="radarGridFactory" class="com.raytheon.uf.common.dataplugin.radar.dataaccess.RadarGridFactory" />
<bean factory-bean="dataAccessRegistry" factory-method="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:

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

(continues on next 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");
DateTime[] 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 implementations 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)
Additional data types may be added in the future. To determine what datatypes are supported display the “type hierarchy” associated with the classes


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
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 code to use the new API at this time. Ideally code will be retrofitted in the future to improve ease of maintainability. The plugin specific 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:

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

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

(continues on next page)
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)

- **DateTime[] getAvailableTimes(IDataRequest request)**
- **DateTime[] getAvailableTimes(IDataRequest request, BinOffset binOffset)**
- **IData[] getData(IDataRequest request, DateTime... 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**

  - **DateTime[] getAvailableTimes(R request)** - queries the database and returns the times that match the request. Some factories may not support this (e.g. maps).

  - **DateTime[] 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, DateTime... 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**

---

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

## Grid Parameters

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSPD</td>
<td>10 Metre neutral wind speed over waves</td>
</tr>
<tr>
<td>WDRT</td>
<td>10 Metre Wind Direction Over Waves</td>
</tr>
<tr>
<td>ARI12H1000YR</td>
<td>12H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>ARI12H100YR</td>
<td>12H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>ARI12H10YR</td>
<td>12H Average Recurrence Interval Accumulation 10 Year</td>
</tr>
<tr>
<td>ARI12H1YR</td>
<td>12H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI12H2000YR</td>
<td>12H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI12H25YR</td>
<td>12H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI12H2YR</td>
<td>12H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI12H5000YR</td>
<td>12H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI12H50YR</td>
<td>12H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI12H5YR</td>
<td>12H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>ARI12H1YR</td>
<td>12H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI12H2000YR</td>
<td>12H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI12H25YR</td>
<td>12H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI12H2YR</td>
<td>12H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI12H5000YR</td>
<td>12H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI12H50YR</td>
<td>12H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI12H5YR</td>
<td>12H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP01H</td>
<td>12 hour Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>GaugeInfIndex12H</td>
<td>12 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>FFG12</td>
<td>12-hr flash flood guidance</td>
</tr>
<tr>
<td>FFR12</td>
<td>12-hr flash flood runoff values</td>
</tr>
<tr>
<td>EchoTop18</td>
<td>18 dBZ Echo Top</td>
</tr>
<tr>
<td>ARI1H1000YR</td>
<td>1H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>ARI1H100YR</td>
<td>1H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>ARI1H10YR</td>
<td>1H Average Recurrence Interval Accumulation 10 Year</td>
</tr>
<tr>
<td>ARI1H1YR</td>
<td>1H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI1H2000YR</td>
<td>1H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI1H25YR</td>
<td>1H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI1H2YR</td>
<td>1H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI1H5000YR</td>
<td>1H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI1H50YR</td>
<td>1H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI1H5YR</td>
<td>1H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP01H</td>
<td>1 hour Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>GaugeInfIndex01H</td>
<td>1 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>QPEFFG01H</td>
<td>1 hour QPE-to-FFG Ratio</td>
</tr>
<tr>
<td>FFG01</td>
<td>1-hr flash flood guidance</td>
</tr>
<tr>
<td>FFR01</td>
<td>1-hr flash flood runoff values</td>
</tr>
<tr>
<td>QPE01</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_ACR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_ALR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_FWR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_KRF</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_MSR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_ORN</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_PTR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_RHA</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_RSA</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_STRA</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_TAR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>QPE01_TIR</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE01_TUA</td>
<td>1-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>EVEC1</td>
<td>1st Vector Component of Electric Field</td>
</tr>
<tr>
<td>BVEC1</td>
<td>1st Vector Component of Magnetic Field</td>
</tr>
<tr>
<td>VEL1</td>
<td>1st Vector Component of Velocity (Coordinate system dependent)</td>
</tr>
<tr>
<td>TCSR1G20</td>
<td>20% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>ARI24H1000YR</td>
<td>24H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>ARI24H100YR</td>
<td>24H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>ARI24H1YR</td>
<td>24H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI24H200YR</td>
<td>24H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI24H25YR</td>
<td>24H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI24H2YR</td>
<td>24H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI24H500YR</td>
<td>24H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI24H50YR</td>
<td>24H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI24H5YR</td>
<td>24H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP24H</td>
<td>24 hour Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>GaugeInfIndex24H</td>
<td>24 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>FFG24</td>
<td>24-hr flash flood guidance</td>
</tr>
<tr>
<td>FFR24</td>
<td>24-hr flash flood runoff values</td>
</tr>
<tr>
<td>QPE24</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_ACR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_ALR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_FWR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_KRF</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_MSR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_ORN</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_PTR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_RHA</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_RSA</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_STR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_TAR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE24_TIR</td>
<td>24-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPF24</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ACR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ALR 205</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_FWR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_KRF</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_MSR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ORN</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_PTR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_RHA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_RSA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_STR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TAR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TIR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TUA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>ARI24H1000YR</td>
<td>24H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>ARI24H100YR</td>
<td>24H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>ARI24H1YR</td>
<td>24H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI24H200YR</td>
<td>24H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI24H25YR</td>
<td>24H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI24H2YR</td>
<td>24H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI24H500YR</td>
<td>24H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI24H50YR</td>
<td>24H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI24H5YR</td>
<td>24H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>QPF24</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ACR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ALR 205</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_FWR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_KRF</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_MSR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_ORN</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_PTR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_RHA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_RSA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_STR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TAR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TIR</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF24_TUA</td>
<td>24-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>ARI24H1000YR</td>
<td>24H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AR2H100YR</td>
<td>2H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>AR2H10YR</td>
<td>2H Average Recurrence Interval Accumulation 10 Year</td>
</tr>
<tr>
<td>AR2H1YR</td>
<td>2H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>AR2H200YR</td>
<td>2H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>AR2H25YR</td>
<td>2H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>AR2H2YR</td>
<td>2H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>AR2H500YR</td>
<td>2H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>AR2H50YR</td>
<td>2H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>AR2H5YR</td>
<td>2H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>EVEC2</td>
<td>2nd Vector Component of Electric Field</td>
</tr>
<tr>
<td>BVEC2</td>
<td>2nd Vector Component of Magnetic Field</td>
</tr>
<tr>
<td>VEL2</td>
<td>2nd Vector Component of Velocity (Coordinate system dependent)</td>
</tr>
<tr>
<td>EchoTop30</td>
<td>30 dBZ Echo Top</td>
</tr>
<tr>
<td>AR30M1000YR</td>
<td>30M Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>AR30M100YR</td>
<td>30M Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>AR30M10YR</td>
<td>30M Average Recurrence Interval Accumulation 10 Year</td>
</tr>
<tr>
<td>AR30M1YR</td>
<td>30M Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>AR30M200YR</td>
<td>30M Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>AR30M25YR</td>
<td>30M Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>AR30M2YR</td>
<td>30M Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>AR30M500YR</td>
<td>30M Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>AR30M50YR</td>
<td>30M Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>AR30M5YR</td>
<td>30M Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP30min</td>
<td>30 min Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>TCSRNG30</td>
<td>30% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>SALIN</td>
<td>3-D Salinity</td>
</tr>
<tr>
<td>WTMPC</td>
<td>3-D Temperature</td>
</tr>
<tr>
<td>AR3H1000YR</td>
<td>3H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>AR3H100YR</td>
<td>3H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>AR3H1YR</td>
<td>3H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>AR3H200YR</td>
<td>3H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>AR3H25YR</td>
<td>3H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>AR3H2YR</td>
<td>3H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>AR3H500YR</td>
<td>3H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>AR3H50YR</td>
<td>3H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>AR3H5YR</td>
<td>3H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP03H</td>
<td>3 hour Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>GaugeInfIndex03H</td>
<td>3 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>QPEFFG03H</td>
<td>3 hour QPE-to-FFG Ratio</td>
</tr>
<tr>
<td>FFG03</td>
<td>3-hr flash flood guidance</td>
</tr>
<tr>
<td>FFR03</td>
<td>3-hr flash flood runoff values</td>
</tr>
<tr>
<td>TSLSA</td>
<td>3-hr pressure tendency (Std. Atmos. Reduction)</td>
</tr>
<tr>
<td>EVEC3</td>
<td>3rd Vector Component of Electric Field</td>
</tr>
<tr>
<td>BVEC3</td>
<td>3rd Vector Component of Magnetic Field</td>
</tr>
<tr>
<td>VEL3</td>
<td>3rd Vector Component of Velocity (Coordinate system dependent)</td>
</tr>
<tr>
<td>TCSRG40</td>
<td>40% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>GaugeInfIndex48H</td>
<td>48 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>EchoTop50</td>
<td>50 dBZ Echo Top</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TCSRG50</td>
<td>50% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>5WAVA</td>
<td>5-wave geopotential height anomaly</td>
</tr>
<tr>
<td>5WAVA</td>
<td>5-Wave Geopotential Height Anomaly</td>
</tr>
<tr>
<td>5WAVH</td>
<td>5-wave geopotential height</td>
</tr>
<tr>
<td>5WAVH</td>
<td>5-Wave Geopotential Height</td>
</tr>
<tr>
<td>EchoTop60</td>
<td>60 dBZ Echo Top</td>
</tr>
<tr>
<td>TCSRG60</td>
<td>60% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>ARI6H1000YR</td>
<td>6H Average Recurrence Interval Accumulation 1000 Year</td>
</tr>
<tr>
<td>ARI6H100YR</td>
<td>6H Average Recurrence Interval Accumulation 100 Year</td>
</tr>
<tr>
<td>ARI6H10YR</td>
<td>6H Average Recurrence Interval Accumulation 10 Year</td>
</tr>
<tr>
<td>ARI6H1YR</td>
<td>6H Average Recurrence Interval Accumulation 1 Year</td>
</tr>
<tr>
<td>ARI6H200YR</td>
<td>6H Average Recurrence Interval Accumulation 200 Year</td>
</tr>
<tr>
<td>ARI6H25YR</td>
<td>6H Average Recurrence Interval Accumulation 25 Year</td>
</tr>
<tr>
<td>ARI6H2YR</td>
<td>6H Average Recurrence Interval Accumulation 2 Year</td>
</tr>
<tr>
<td>ARI6H500YR</td>
<td>6H Average Recurrence Interval Accumulation 500 Year</td>
</tr>
<tr>
<td>ARI6H50YR</td>
<td>6H Average Recurrence Interval Accumulation 50 Year</td>
</tr>
<tr>
<td>ARI6H5YR</td>
<td>6H Average Recurrence Interval Accumulation 5 Year</td>
</tr>
<tr>
<td>PRP06H</td>
<td>6 hour Precipitation Accumulation Return Period</td>
</tr>
<tr>
<td>GaugeInflIndex06H</td>
<td>6 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>QPEFFG06H</td>
<td>6 hour QPE-to-FFG Ratio</td>
</tr>
<tr>
<td>FFG06</td>
<td>6-hr flash flood guidance</td>
</tr>
<tr>
<td>FFR06</td>
<td>6-hr flash flood runoff values</td>
</tr>
<tr>
<td>QPE06</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_ACR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_ALR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_FWR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_KRF</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_MSR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_ORN</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_PTR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_RHA</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_RSA</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_STR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_TAR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_TIR</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPE06_TUA</td>
<td>6-hr Quantitative Precip Estimate</td>
</tr>
<tr>
<td>QPF06</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_ACR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_ALR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_FWR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_KRF</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_MSR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_ORN</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_PTR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_RHA</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_RSA</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_STR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_TAR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>QPF06_TIR</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>QPF06_TUA</td>
<td>6-hr Quantitative Precip Forecast</td>
</tr>
<tr>
<td>TCSRG70</td>
<td>70% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>GaugeInIndex72H</td>
<td>72 hour QPE Gauge Influence Index</td>
</tr>
<tr>
<td>TCSRG80</td>
<td>80% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>TCSRG90</td>
<td>90% Tropical Cyclone Storm Surge Exceedance</td>
</tr>
<tr>
<td>ABSD</td>
<td>Absolute divergence</td>
</tr>
<tr>
<td>ABSH</td>
<td>Absolute Humidity</td>
</tr>
<tr>
<td>AH</td>
<td>Absolute humidity</td>
</tr>
<tr>
<td>ABSV</td>
<td>Absolute vorticity</td>
</tr>
<tr>
<td>ASD</td>
<td>Accumulated Snow Depth</td>
</tr>
<tr>
<td>ACOND</td>
<td>Aerodynamic conductance</td>
</tr>
<tr>
<td>AETY</td>
<td>Aerosol type</td>
</tr>
<tr>
<td>AC137</td>
<td>Air concentration of Caesium 137</td>
</tr>
<tr>
<td>AI131</td>
<td>Air concentration of Iodine 131</td>
</tr>
<tr>
<td>ARADP</td>
<td>Air concentration of radioactive pollutant</td>
</tr>
<tr>
<td>ALBDO</td>
<td>Albedo</td>
</tr>
<tr>
<td>ACWVH</td>
<td>Altimeter corrected wave height</td>
</tr>
<tr>
<td>ALRRRC</td>
<td>Altimeter Range Relative Correction</td>
</tr>
<tr>
<td>ASET</td>
<td>Altimeter setting</td>
</tr>
<tr>
<td>AWVH</td>
<td>Altimeter wave height</td>
</tr>
<tr>
<td>ALTMSL</td>
<td>Altitude above mean sea level</td>
</tr>
<tr>
<td>ANCConectiveOutlook</td>
<td>ANC Convective Outlook</td>
</tr>
<tr>
<td>ANCFinalForecast</td>
<td>ANC Final Forecast</td>
</tr>
<tr>
<td>AOSGSO</td>
<td>Angle Of Sub-Grid Scale Orography</td>
</tr>
<tr>
<td>ASGSO</td>
<td>Anisotropy Of Sub-Grid Scale Orography</td>
</tr>
<tr>
<td>APTMP</td>
<td>Apparent Temperature</td>
</tr>
<tr>
<td>ARBTXT</td>
<td>Arbitrary text string</td>
</tr>
<tr>
<td>ASHFL</td>
<td>Assimilative Heat Flux</td>
</tr>
<tr>
<td>AMIXL</td>
<td>Asymptotic mixing length scale</td>
</tr>
<tr>
<td>ATMDIV</td>
<td>Atmospheric Divergence</td>
</tr>
<tr>
<td>AVSFT</td>
<td>Average surface skin temperature</td>
</tr>
<tr>
<td>BARET</td>
<td>Bare soil surface skin temperature</td>
</tr>
<tr>
<td>BKENG</td>
<td>Barotropic Kinetic Energy</td>
</tr>
<tr>
<td>UBARO</td>
<td>Barotropic U velocity</td>
</tr>
<tr>
<td>UBARO</td>
<td>Barotropic U Velocity</td>
</tr>
<tr>
<td>VBARO</td>
<td>Barotropic V velocity</td>
</tr>
<tr>
<td>VBARO</td>
<td>Barotropic V Velocity</td>
</tr>
<tr>
<td>BGRUN</td>
<td>Baseflow-groundwater runoff</td>
</tr>
<tr>
<td>BASR</td>
<td>Base reflectivity</td>
</tr>
<tr>
<td>BASRW</td>
<td>Base spectrum width</td>
</tr>
<tr>
<td>4LFTX</td>
<td>Best (4-layer) lifted index</td>
</tr>
<tr>
<td>4LFTX</td>
<td>Best (4 layer) Lifted Index</td>
</tr>
<tr>
<td>BLI</td>
<td>Best lifted index (to 500 mb)</td>
</tr>
<tr>
<td>BMIXL</td>
<td>Blackadars mixing length scale</td>
</tr>
<tr>
<td>BLST</td>
<td>Bottom layer soil temperature</td>
</tr>
<tr>
<td>NONE</td>
<td>Bottom of Ocean Isothermal Layer</td>
</tr>
<tr>
<td>OBIL</td>
<td>Bottom of Ocean Isothermal Layer</td>
</tr>
<tr>
<td>NONE</td>
<td>Bottom of Ocean Mixed Layer (m)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>OBML</td>
<td>Bottom of Ocean Mixed Layer (m)</td>
</tr>
<tr>
<td>BCBL</td>
<td>Boundary layer cloud bottom level</td>
</tr>
<tr>
<td>BCBL</td>
<td>Boundary Layer Cloud Bottom Level</td>
</tr>
<tr>
<td>BCLY</td>
<td>Boundary Layer Cloud Layer</td>
</tr>
<tr>
<td>BCY</td>
<td>Boundary layer cloud layer</td>
</tr>
<tr>
<td>BCTL</td>
<td>Boundary layer cloud top level</td>
</tr>
<tr>
<td>BLYSP</td>
<td>Boundary layer dissipation</td>
</tr>
<tr>
<td>BrightBandBottomHeight</td>
<td>Bright Band Bottom Height</td>
</tr>
<tr>
<td>BrightBandTopHeight</td>
<td>Bright Band Top Height</td>
</tr>
<tr>
<td>BRTMP</td>
<td>Brightness temperature</td>
</tr>
<tr>
<td>CAIIRAD</td>
<td>CaII-K Radiance</td>
</tr>
<tr>
<td>CCOND</td>
<td>Canopy conductance</td>
</tr>
<tr>
<td>EVCW</td>
<td>Canopy water evaporation</td>
</tr>
<tr>
<td>CONVP</td>
<td>Categorical Convection</td>
</tr>
<tr>
<td>CFRZR</td>
<td>Categorical Freezing Rain</td>
</tr>
<tr>
<td>CFRZR</td>
<td>Categorical Freezing Rain</td>
</tr>
<tr>
<td>CFRZR</td>
<td>Categorical Freezing Rain (yes=1; no=0)</td>
</tr>
<tr>
<td>CFRZR</td>
<td>Categorical Freezing Rain (yes=1; no=0)</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical ice pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical ice pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CICEP</td>
<td>Categorical Ice Pellets</td>
</tr>
<tr>
<td>CLGTN</td>
<td>Categorical Lightning</td>
</tr>
<tr>
<td>OZCAT</td>
<td>Categorical Ozone Concentration</td>
</tr>
<tr>
<td>OZCAT</td>
<td>Categorical Ozone Concentration</td>
</tr>
<tr>
<td>CRAIN</td>
<td>Categorical Rain</td>
</tr>
<tr>
<td>CRAIN</td>
<td>Categorical Rain</td>
</tr>
<tr>
<td>CRAIN</td>
<td>Categorical rain</td>
</tr>
<tr>
<td>CRAIN</td>
<td>Categorical Rain (yes=1; no=0)</td>
</tr>
<tr>
<td>CRAIN</td>
<td>Categorical Rain (yes=1; no=0)</td>
</tr>
<tr>
<td>SVRTS</td>
<td>Categorical Severe Thunderstorm</td>
</tr>
<tr>
<td>SVRTS</td>
<td>Categorical Severe Thunderstorm</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical Snow</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical Snow</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical Snow</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical snow</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical Snow (yes=1; no=0)</td>
</tr>
<tr>
<td>CSNOW</td>
<td>Categorical Snow (yes=1; no=0)</td>
</tr>
<tr>
<td>CTSTM</td>
<td>Categorical Thunderstorm (1=yes, 0-no)</td>
</tr>
<tr>
<td>TSTM</td>
<td>Categorical Thunderstorm (1=yes, 0-no)</td>
</tr>
<tr>
<td>CCEIL</td>
<td>Ceiling</td>
</tr>
<tr>
<td>LightningDensity15min</td>
<td>CG Lightning Density (15 min.)</td>
</tr>
</tbody>
</table>

Table 1 – continued from previous page
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightningDensity1min</td>
<td>CG Lightning Density (1 min.)</td>
</tr>
<tr>
<td>LightningDensity30min</td>
<td>CG Lightning Density (30 min.)</td>
</tr>
<tr>
<td>LightningDensity5min</td>
<td>CG Lightning Density (5 min.)</td>
</tr>
<tr>
<td>LightningProbabilityNext30min</td>
<td>CG Lightning Probability (0-30 min.)</td>
</tr>
<tr>
<td>CAT</td>
<td>Clear Air Turbulence (CAT)</td>
</tr>
<tr>
<td>CAT</td>
<td>Clear Air Turbulence (CAT)</td>
</tr>
<tr>
<td>CSDLF</td>
<td>Clear Sky Downward Long Wave Flux</td>
</tr>
<tr>
<td>CSDF</td>
<td>Clear sky downward solar flux</td>
</tr>
<tr>
<td>CSULF</td>
<td>Clear Sky Upward Long Wave Flux</td>
</tr>
<tr>
<td>CSUSF</td>
<td>Clear Sky Upward Solar Flux</td>
</tr>
<tr>
<td>CDUVB</td>
<td>Clear sky UV-B downward solar flux</td>
</tr>
<tr>
<td>CAMT</td>
<td>Cloud amount</td>
</tr>
<tr>
<td>CBL</td>
<td>Cloud Base Level</td>
</tr>
<tr>
<td>CBASE</td>
<td>Cloud base</td>
</tr>
<tr>
<td>CEIL</td>
<td>Cloud Ceiling</td>
</tr>
<tr>
<td>CLG</td>
<td>Cloud ceiling</td>
</tr>
<tr>
<td>CLG</td>
<td>Cloud Ceiling</td>
</tr>
<tr>
<td>CloudCover</td>
<td>Cloud Cover</td>
</tr>
<tr>
<td>CFNLFLF</td>
<td>Cloud Forcing Net Long Wave Flux</td>
</tr>
<tr>
<td>CFNSF</td>
<td>Cloud Forcing Net Solar Flux</td>
</tr>
<tr>
<td>CDCIMR</td>
<td>Cloud Ice Mixing Ratio</td>
</tr>
<tr>
<td>CICE</td>
<td>Cloud ice</td>
</tr>
<tr>
<td>CLOUDDM</td>
<td>Cloud mask</td>
</tr>
<tr>
<td>CLWMR</td>
<td>Cloud Mixing Ratio</td>
</tr>
<tr>
<td>CLWMR</td>
<td>Cloud mixing ratio</td>
</tr>
<tr>
<td>CTOPHQLI</td>
<td>Cloud top height quality indicator</td>
</tr>
<tr>
<td>CTOP</td>
<td>Cloud top</td>
</tr>
<tr>
<td>heightCTHGT</td>
<td>Cloud top height</td>
</tr>
<tr>
<td>CTYP</td>
<td>Cloud type</td>
</tr>
<tr>
<td>CWAT</td>
<td>Cloud water</td>
</tr>
<tr>
<td>CWORK</td>
<td>Cloud work function</td>
</tr>
<tr>
<td>CWORK</td>
<td>Cloud Work Function</td>
</tr>
<tr>
<td>CDWW</td>
<td>Coefficient of Drag With Waves</td>
</tr>
<tr>
<td>CISOILW</td>
<td>Column-Integrated Soil Water</td>
</tr>
<tr>
<td>RFC</td>
<td>Composite reflectivity</td>
</tr>
<tr>
<td>MergedReflectivityQCComposite</td>
<td>Composite Reflectivity</td>
</tr>
<tr>
<td>HeighCompositeReflectivity</td>
<td>Composite Reflectivity Height</td>
</tr>
<tr>
<td>MergedReflectivityQComposite</td>
<td>Composite Reflectivity Mosaic</td>
</tr>
<tr>
<td>EF25M</td>
<td>Cond 25% pcpn smy fractile past 24 hrs</td>
</tr>
<tr>
<td>EF50M</td>
<td>Cond 50% pcpn smy fractile past 24 hrs</td>
</tr>
<tr>
<td>TCOND</td>
<td>Condensate</td>
</tr>
<tr>
<td>CONDE</td>
<td>Condensate</td>
</tr>
<tr>
<td>CONDP</td>
<td>Condensation Pressure of Parcali Lifted From Indicate Surface</td>
</tr>
<tr>
<td>CONDP</td>
<td>Condensation Pressure of Parcal Lifted From Indicate Surface</td>
</tr>
<tr>
<td>CPPAF</td>
<td>Conditional percent precipitation amount fractile for an overall period (Encoded as an accumulation)</td>
</tr>
<tr>
<td>CICEL</td>
<td>Confidence - Ceiling</td>
</tr>
<tr>
<td>CIFLT</td>
<td>Confidence - Flight Category</td>
</tr>
<tr>
<td>CIVIS</td>
<td>Confidence - Visibility</td>
</tr>
<tr>
<td>CONTB</td>
<td>Contrail base</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>CONTB</td>
<td>Contrail Base</td>
</tr>
<tr>
<td>CONTE</td>
<td>Contrail engine type</td>
</tr>
<tr>
<td>CONTET</td>
<td>Contrail Engine Type</td>
</tr>
<tr>
<td>CONTI</td>
<td>Contrail intensity</td>
</tr>
<tr>
<td>CONTI</td>
<td>Contrail Intensity</td>
</tr>
<tr>
<td>CONTT</td>
<td>Contrail top</td>
</tr>
<tr>
<td>CONUSMergedReflectivity</td>
<td>CONUS Merged Reflectivity</td>
</tr>
<tr>
<td>CONUSPlusMergedReflectivity</td>
<td>CONUS-Plus Merged Reflectivity</td>
</tr>
<tr>
<td>CONVP</td>
<td>Convection Potential</td>
</tr>
<tr>
<td>CAPE</td>
<td>Convective available potential energy</td>
</tr>
<tr>
<td>CCBL</td>
<td>Convective cloud bottom level</td>
</tr>
<tr>
<td>CCBL</td>
<td>Convective Cloud Bottom Level</td>
</tr>
<tr>
<td>CCLY</td>
<td>Convective Cloud</td>
</tr>
<tr>
<td>CCY</td>
<td>Convective cloud</td>
</tr>
<tr>
<td>CDCON</td>
<td>Convective cloud cover</td>
</tr>
<tr>
<td>CUEFI</td>
<td>Convective Cloud Efficiency</td>
</tr>
<tr>
<td>CUEFI</td>
<td>Convective cloud efficiency</td>
</tr>
<tr>
<td>MFLUX</td>
<td>Convective Cloud Mass Flux</td>
</tr>
<tr>
<td>CCTL</td>
<td>Convective cloud top level</td>
</tr>
<tr>
<td>CCTL</td>
<td>Convective Cloud Top Level</td>
</tr>
<tr>
<td>CNVDEMF</td>
<td>Convective detrainment mass flux</td>
</tr>
<tr>
<td>CNVDMF</td>
<td>Convective downdraft mass flux</td>
</tr>
<tr>
<td>CNGWDV</td>
<td>Convective Gravity wave drag meridional acceleration</td>
</tr>
<tr>
<td>CNGWDU</td>
<td>Convective Gravity wave drag zonal acceleration</td>
</tr>
<tr>
<td>CIN</td>
<td>Convective inhibition</td>
</tr>
<tr>
<td>CNVV</td>
<td>Convective meridional momentum mixing acceleration</td>
</tr>
<tr>
<td>ACPCP</td>
<td>Convective Precipitation</td>
</tr>
<tr>
<td>ACPCP</td>
<td>Convective precipitation</td>
</tr>
<tr>
<td>ACPCPN</td>
<td>Convective precipitation (nearest grid point)</td>
</tr>
<tr>
<td>ACPCPN</td>
<td>Convective precipitation (nearest grid point)</td>
</tr>
<tr>
<td>CPRAT</td>
<td>Convective Precipitation Rate</td>
</tr>
<tr>
<td>CPRAT</td>
<td>Convective Precipitation Rate</td>
</tr>
<tr>
<td>CPRAT</td>
<td>Convective precipitation rate</td>
</tr>
<tr>
<td>CSRATE</td>
<td>Convective Snowfall Rate</td>
</tr>
<tr>
<td>CSRATE</td>
<td>Convective Snowfall Rate</td>
</tr>
<tr>
<td>CSRWE</td>
<td>Convective Snowfall Rate Water Equivalent</td>
</tr>
<tr>
<td>CSRWE</td>
<td>Convective Snowfall Rate Water Equivalent</td>
</tr>
<tr>
<td>SNOC</td>
<td>Convective snow</td>
</tr>
<tr>
<td>CNVUMF</td>
<td>Convective updraft mass flux</td>
</tr>
<tr>
<td>CWP</td>
<td>Convective Water Precipitation</td>
</tr>
<tr>
<td>CWP</td>
<td>Convective Water Precipitation</td>
</tr>
<tr>
<td>CWDI</td>
<td>Convective Weather Detection Index</td>
</tr>
<tr>
<td>CNVU</td>
<td>Convective zonal momentum mixing acceleration</td>
</tr>
<tr>
<td>SNO C</td>
<td>Convect Snow</td>
</tr>
<tr>
<td>NTRNFLUX</td>
<td>Cosmic Ray Neutron Flux</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVTZ</td>
<td>Covariance between izonal component of the wind and temperature. Defined as ([uT]-[u][T]), where &quot;[&quot; indicates the mean over the indicated time span.</td>
</tr>
<tr>
<td>COVMM</td>
<td>Covariance between meridional and meridional components of the wind. Defined as ([vv]-[v][v]).</td>
</tr>
<tr>
<td>COVMZ</td>
<td>Covariance between Meridional and Zonal Components of the wind.</td>
</tr>
<tr>
<td>COVTM</td>
<td>Covariance between meridional component of the wind and temperature. Defined as ([vT]-[v][T]).</td>
</tr>
<tr>
<td>COVQM</td>
<td>Covariance between specific humidity and meridional components of the wind. Defined as ([vq]-[v][q]).</td>
</tr>
<tr>
<td>COVQQ</td>
<td>Covariance between specific humidity and specific humidity. Defined as ([qq]-[q][q]), where &quot;[]&quot; indicates the mean over the indicated time span.</td>
</tr>
<tr>
<td>COVQVV</td>
<td>Covariance between specific humidity and vertical components of the wind. Defined as ([\Omega q]-[\Omega][q]).</td>
</tr>
<tr>
<td>COVQZ</td>
<td>Covariance between specific humidity and zonal components of the wind. Defined as ([uq]-[u][q]).</td>
</tr>
<tr>
<td>COVPSPS</td>
<td>Covariance between surface pressure and surface pressure. Defined as ([Psfc]-[Psfc][Psfc]), where &quot;[&quot; indicates the mean over the indicated time span.</td>
</tr>
<tr>
<td>COVTM</td>
<td>Covariance between Temperature and Meridional Components of the wind.</td>
</tr>
<tr>
<td>COVTT</td>
<td>Covariance between temperature and temperature. Defined as ([TT]-[T][T]), where &quot;[&quot; indicates the mean over the indicated time span.</td>
</tr>
<tr>
<td>COVTW</td>
<td>Covariance between temperature and vertical component of the wind. Defined as ([wT]-[w][T]).</td>
</tr>
<tr>
<td>COVTVV</td>
<td>Covariance between temperature and vertical components of the wind. Defined as ([\Omega T]-[\Omega][T]).</td>
</tr>
<tr>
<td>COVTZ</td>
<td>Covariance between Temperature and Zonal Components of the wind.</td>
</tr>
<tr>
<td>COVVVVVV</td>
<td>Covariance between vertical and vertical components of the wind. Defined as ([\Omega\Omega]-[\Omega][\Omega]), where &quot;[&quot; indicates the mean over the indicated time span.</td>
</tr>
<tr>
<td>COVMZ</td>
<td>Covariance between zonal and meridional components of the wind. Defined as ([uv]-[u][v]).</td>
</tr>
<tr>
<td>COVZZ</td>
<td>Covariance between zonal and zonal components of the wind. Defined as ([uu]-[u][u]), where &quot;[&quot; indicates the mean over the indicated time span.</td>
</tr>
</tbody>
</table>

CrestMaxStreamflow  
CrestMaxUStreamflow  
CrestSoilMoisture  
CRTFRQ  
CB  
CBHE  
CT  
DIRC  
SPC  
DCBL  
DCBL  
DCCBL  
DCCBL  
DCCTL  
DCCTL  
CNVHR  
CNVMR  
CNVMR  
DALT  
DEN  
DBLL  
DPBLW  
DBSL  
DPMSL  
REFZI  
REFZI  
REFZC  
REFZC  
REFZC  
REFZC  
REFZER  
REFZER  
REFD  
DEVMSL  

Deviation of sea level from mean
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPR</td>
<td>Dew point depression or deficit</td>
</tr>
<tr>
<td>DPT</td>
<td>Dew point temperature</td>
</tr>
<tr>
<td>DIREC</td>
<td>Direct Evaporation Cease(Soil Moisture)</td>
</tr>
<tr>
<td>SMDRY</td>
<td>Direct evaporation cease (soil moisture)</td>
</tr>
<tr>
<td>EVBS</td>
<td>Direct evaporation from bare soil</td>
</tr>
<tr>
<td>DIRWWW</td>
<td>Directional Width of The Wind Waves</td>
</tr>
<tr>
<td>Degree true</td>
<td>Direction Degrees true</td>
</tr>
<tr>
<td>WWSDIR</td>
<td>Direction of combined wind waves and swell</td>
</tr>
<tr>
<td>DICED</td>
<td>Direction of ice drift</td>
</tr>
<tr>
<td>SWDIR</td>
<td>Direction of swell waves</td>
</tr>
<tr>
<td>WVDIR</td>
<td>Direction of wind waves</td>
</tr>
<tr>
<td>DSKDAY</td>
<td>Disk Intensity Day</td>
</tr>
<tr>
<td>DSKINT</td>
<td>Disk Intensity</td>
</tr>
<tr>
<td>DSKNGT</td>
<td>Disk Intensity Night</td>
</tr>
<tr>
<td>DLWRF</td>
<td>Downward Long-Wave Rad. Flux</td>
</tr>
<tr>
<td>DSWRF</td>
<td>Downward Short-Wave Rad. Flux</td>
</tr>
<tr>
<td>DTRF</td>
<td>Downward Total radiation Flux</td>
</tr>
<tr>
<td>DWUVR</td>
<td>Downward UV Radiation</td>
</tr>
<tr>
<td>CD</td>
<td>Drag Coefficient</td>
</tr>
<tr>
<td>ELON</td>
<td>East Longitude (0 - 360)</td>
</tr>
<tr>
<td>ELONN</td>
<td>East Longitude (nearest neighbor) (0 - 360)</td>
</tr>
<tr>
<td>RETOP</td>
<td>Echo Top</td>
</tr>
<tr>
<td>RADT</td>
<td>Effective radiative skin temperature</td>
</tr>
<tr>
<td>ETOT</td>
<td>Electric Field Magnitude</td>
</tr>
<tr>
<td>ELCDEN</td>
<td>Electron Density</td>
</tr>
<tr>
<td>DIFEFLUX</td>
<td>Electron Flux (Differential)</td>
</tr>
<tr>
<td>INTEFLUX</td>
<td>Electron Flux (Integral)</td>
</tr>
<tr>
<td>ELECTMP</td>
<td>Electron Temperature</td>
</tr>
<tr>
<td>ELSCT</td>
<td>Elevation of snow covered terrain</td>
</tr>
<tr>
<td>EHELX</td>
<td>Energy helicity index</td>
</tr>
<tr>
<td>EATM</td>
<td>Entire atmosphere (considered as a single layer)</td>
</tr>
<tr>
<td>EA</td>
<td>Entire Atmosphere</td>
</tr>
<tr>
<td>NONE</td>
<td>Entire Atmosphere</td>
</tr>
<tr>
<td>EOCN</td>
<td>Entire ocean (considered as a single layer)</td>
</tr>
<tr>
<td>NONE</td>
<td>Entire Ocean</td>
</tr>
<tr>
<td>EHLT</td>
<td>Equilibrium level</td>
</tr>
<tr>
<td>EHHT</td>
<td>Equilibrium Level</td>
</tr>
<tr>
<td>REFZC</td>
<td>Equivalent radar reflectivity factor for parameterized convection</td>
</tr>
<tr>
<td>REFZR</td>
<td>Equivalent radar reflectivity factor for rain</td>
</tr>
<tr>
<td>REFZI</td>
<td>Equivalent radar reflectivity factor for snow</td>
</tr>
<tr>
<td>ESTPC</td>
<td>Estimated precipitation</td>
</tr>
<tr>
<td>ESTUWIND</td>
<td>Estimated u component of wind</td>
</tr>
<tr>
<td>ESTVWIND</td>
<td>Estimated v component of wind</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ELYR</td>
<td>Eta Level</td>
</tr>
<tr>
<td>ETAL</td>
<td>Eta Level</td>
</tr>
<tr>
<td>EUVRAD</td>
<td>EUV Radiance</td>
</tr>
<tr>
<td>EVP</td>
<td>Evaporation</td>
</tr>
<tr>
<td>EVP</td>
<td>Evaporation - Precipitation</td>
</tr>
<tr>
<td>EMNP</td>
<td>Evaporation - Precipitation</td>
</tr>
<tr>
<td>EMNP</td>
<td>Evaporation - Precipitation</td>
</tr>
<tr>
<td>EVAP</td>
<td>Evapotranspiration</td>
</tr>
<tr>
<td>SFEXC</td>
<td>Exchange coefficient</td>
</tr>
<tr>
<td>SFEXC</td>
<td>Exchange coefficient</td>
</tr>
<tr>
<td>ETCLWL</td>
<td>Extra Tropical Storm Surge Combined Surge and Tide</td>
</tr>
<tr>
<td>ETSRG</td>
<td>Extra Tropical Storm Surge</td>
</tr>
<tr>
<td>F107</td>
<td>F10.7</td>
</tr>
<tr>
<td>FLDCP</td>
<td>Field Capacity</td>
</tr>
<tr>
<td>FIREDI</td>
<td>Fire Detection Indicator</td>
</tr>
<tr>
<td>FIREODT</td>
<td>Fire Outlook Due to Dry Thunderstorm</td>
</tr>
<tr>
<td>FIREOLK</td>
<td>Fire Outlook</td>
</tr>
<tr>
<td>FFG</td>
<td>Flash flood guidance (Encoded as an accumulation over a floating subinterval of time between two forecast times)</td>
</tr>
<tr>
<td>FFRUN</td>
<td>Flash flood runoff (Encoded as an accumulation over a floating subinterval of time)</td>
</tr>
<tr>
<td>FLGHT</td>
<td>Flight Category</td>
</tr>
<tr>
<td>QREC</td>
<td>Flood plain recharge</td>
</tr>
<tr>
<td>ModelHeight0C</td>
<td>Freezing Level Height</td>
</tr>
<tr>
<td>FRZR</td>
<td>Freezing Rain</td>
</tr>
<tr>
<td>FRZR</td>
<td>Freezing Rain</td>
</tr>
<tr>
<td>FPRATE</td>
<td>Freezing Rain Precipitation Rate</td>
</tr>
<tr>
<td>FPRATE</td>
<td>Freezing Rain Precipitation Rate</td>
</tr>
<tr>
<td>FREQ</td>
<td>Frequency</td>
</tr>
<tr>
<td>FRICV</td>
<td>Frictional velocity</td>
</tr>
<tr>
<td>FRICV</td>
<td>Frictional Velocity</td>
</tr>
<tr>
<td>FRICV</td>
<td>Friction Velocity</td>
</tr>
<tr>
<td>FROZR</td>
<td>Frozen Rain</td>
</tr>
<tr>
<td>FROZR</td>
<td>Frozen Rain</td>
</tr>
<tr>
<td>GHT</td>
<td>Geometrical height</td>
</tr>
<tr>
<td>DBSS</td>
<td>Geometric Depth Below Sea Surface</td>
</tr>
<tr>
<td>DIST</td>
<td>Geometric height</td>
</tr>
<tr>
<td>GPA</td>
<td>Geopotential height anomaly</td>
</tr>
<tr>
<td>GH</td>
<td>Geopotential height</td>
</tr>
<tr>
<td>HGT</td>
<td>Geopotential height</td>
</tr>
<tr>
<td>HGTN</td>
<td>Geopotential Height (nearest grid point)</td>
</tr>
<tr>
<td>GP</td>
<td>Geopotential</td>
</tr>
<tr>
<td>GRAD</td>
<td>Global radiation flux</td>
</tr>
<tr>
<td>GRAUP</td>
<td>Graupel (snow pellets)</td>
</tr>
<tr>
<td>GRLE</td>
<td>Grauple</td>
</tr>
<tr>
<td>GSGSO</td>
<td>Gravity Of Sub-Grid Scale Orography</td>
</tr>
<tr>
<td>GWDV</td>
<td>Gravity wave drag meridional acceleration</td>
</tr>
<tr>
<td>GWDU</td>
<td>Gravity wave drag zonal acceleration</td>
</tr>
<tr>
<td>GCBL</td>
<td>Grid scale cloud bottom level</td>
</tr>
<tr>
<td>GCBL</td>
<td>Grid Scale Cloud Bottom Level</td>
</tr>
<tr>
<td>GSCBL</td>
<td>Grid Scale Cloud Bottom Level</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>GCTL</td>
<td>Grid scale cloud top level</td>
</tr>
<tr>
<td>GC137</td>
<td>Ground deposition of Caesium 137</td>
</tr>
<tr>
<td>GI131</td>
<td>Ground deposition of Iodine 131</td>
</tr>
<tr>
<td>GRADP</td>
<td>Ground deposition of radioactive</td>
</tr>
<tr>
<td>GFLUX</td>
<td>Ground heat flux</td>
</tr>
<tr>
<td>SFC</td>
<td>Ground or Water Surface</td>
</tr>
<tr>
<td>GWREC</td>
<td>Groundwater recharge</td>
</tr>
<tr>
<td>HAIL</td>
<td>Hail</td>
</tr>
<tr>
<td>HAILPROB</td>
<td>Hail probability</td>
</tr>
<tr>
<td>HINDEX</td>
<td>Haines Index</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>Hall Conductivity</td>
</tr>
<tr>
<td>HARAD</td>
<td>H-Alpha Radiance</td>
</tr>
<tr>
<td>HELCOR</td>
<td>Heliospheric Radiance</td>
</tr>
<tr>
<td>ABSRF</td>
<td>HF Absorption</td>
</tr>
<tr>
<td>ABSFRQ</td>
<td>HF Absorption Frequency</td>
</tr>
<tr>
<td>HPRIMF</td>
<td>h’F</td>
</tr>
<tr>
<td>HCBL</td>
<td>High cloud bottom level</td>
</tr>
<tr>
<td>HCDC</td>
<td>High cloud cover</td>
</tr>
<tr>
<td>HCL</td>
<td>High cloud layer</td>
</tr>
<tr>
<td>H50Above0C</td>
<td>Height of 50 dBZ Echo Above 0C</td>
</tr>
<tr>
<td>H50AboveM20C</td>
<td>Height of 50 dBZ Echo Above -20C</td>
</tr>
<tr>
<td>H60Above0C</td>
<td>Height of 60 dBZ Echo Above 0C</td>
</tr>
<tr>
<td>H60AboveM20C</td>
<td>Height of 60 dBZ Echo Above -20C</td>
</tr>
<tr>
<td>HSCLW</td>
<td>Highest top level of supercooled liquid water layer</td>
</tr>
<tr>
<td>HSCLW</td>
<td>Highest Top Level of Supercooled Liquid Water Layer</td>
</tr>
<tr>
<td>HTFL</td>
<td>Highest tropospheric freezing level</td>
</tr>
<tr>
<td>HTFL</td>
<td>Highest Tropospheric Freezing Level</td>
</tr>
<tr>
<td>HighLayerCompositeReflectivity</td>
<td>High Layer Composite Reflectivity (24-60 kft)</td>
</tr>
<tr>
<td>HAVNI</td>
<td>High-Level aviation interest</td>
</tr>
<tr>
<td>HRCONO</td>
<td>High risk convective outlook</td>
</tr>
<tr>
<td>MCONV</td>
<td>Horizontal Moisture Convergence</td>
</tr>
<tr>
<td>MCONV</td>
<td>Horizontal moisture convergence</td>
</tr>
<tr>
<td>MCONV</td>
<td>Horizontal Moisture Divergence</td>
</tr>
<tr>
<td>MCONV</td>
<td>Horizontal moisture divergence</td>
</tr>
<tr>
<td>MCONV</td>
<td>Horizontal Moisture Divergence</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLX</td>
<td>Horizontal momentum flux</td>
</tr>
<tr>
<td>MFLX</td>
<td>Horizontal Momentum Flux</td>
</tr>
<tr>
<td>MFLX</td>
<td>Horizontal Momentum Flux</td>
</tr>
<tr>
<td>CompositeReflectivityMaxHourly</td>
<td>Hourly Composite Reflectivity Maximum</td>
</tr>
<tr>
<td>MAXDVV</td>
<td>Hourly Maximum of Downward Vertical Vorticity in the lowest 400hPa</td>
</tr>
<tr>
<td>MAXREH</td>
<td>Hourly Maximum of Simulated Reflectivity at 1 km AGL</td>
</tr>
<tr>
<td>MXUPHL</td>
<td>Hourly Maximum of Updraft Helicity over Layer 2-5 km AGL</td>
</tr>
<tr>
<td>MXUPHL</td>
<td>Hourly Maximum of Updraft Helicity over Layer 2km to 5 km AGL</td>
</tr>
<tr>
<td>MAXUVVV</td>
<td>Hourly Maximum of Upward Vertical Vorticity in the lowest 400hPa</td>
</tr>
<tr>
<td>MIXR</td>
<td>Humidity Mixing Ratio</td>
</tr>
<tr>
<td>MIXR</td>
<td>Humidity mixing ratio</td>
</tr>
<tr>
<td>RCQ</td>
<td>Humidity parameter in canopy conductance</td>
</tr>
<tr>
<td>RCQ</td>
<td>Humidity parameter in canopy conductance</td>
</tr>
<tr>
<td>HYBL</td>
<td>Hybrid Level</td>
</tr>
<tr>
<td>HCBB</td>
<td>ICAO Height at Cumulonimbus Bas</td>
</tr>
<tr>
<td>HCBT</td>
<td>ICAO Height at Cumulonimbus Top</td>
</tr>
<tr>
<td>HECBB</td>
<td>ICAO Height at Embedded Cumulonimbus Bas</td>
</tr>
<tr>
<td>HECBT</td>
<td>ICAO Height at Embedded Cumulonimbus To</td>
</tr>
<tr>
<td>ICAHT</td>
<td>ICAO Standard Atmosphere Reference Height</td>
</tr>
<tr>
<td>ICEC</td>
<td>Ice cover</td>
</tr>
<tr>
<td>ICED</td>
<td>Ice divergence</td>
</tr>
<tr>
<td>FICE</td>
<td>Ice fraction of total condensate</td>
</tr>
<tr>
<td>FICE</td>
<td>Ice fraction of total condensate</td>
</tr>
<tr>
<td>FICE</td>
<td>Ice fraction of total condensate</td>
</tr>
<tr>
<td>surface</td>
<td>ice-free water</td>
</tr>
<tr>
<td>ICWAT</td>
<td>ice-free water surface</td>
</tr>
<tr>
<td>ICEG</td>
<td>Ice growth rate</td>
</tr>
<tr>
<td>IPRATE</td>
<td>Ice Pellets Precipitation Rate</td>
</tr>
<tr>
<td>IPRATE</td>
<td>Ice Pellets Precipitation Rate</td>
</tr>
<tr>
<td>ICE T</td>
<td>Ice Temperature</td>
</tr>
<tr>
<td>ICETK</td>
<td>Ice thickness</td>
</tr>
<tr>
<td>ICMR</td>
<td>Ice Water Mixing Ratio</td>
</tr>
<tr>
<td>ICMR</td>
<td>Ice water mixing ratio</td>
</tr>
<tr>
<td>ICIB</td>
<td>Icing base</td>
</tr>
<tr>
<td>ICIB</td>
<td>Icing Base</td>
</tr>
<tr>
<td>ICIP</td>
<td>Icing</td>
</tr>
<tr>
<td>ICIP</td>
<td>Icing Potential</td>
</tr>
<tr>
<td>TIPE</td>
<td>Icing Potential</td>
</tr>
<tr>
<td>ICPRB</td>
<td>Icing probability</td>
</tr>
<tr>
<td>ICI</td>
<td>Icing Icing Severity</td>
</tr>
<tr>
<td>ICSEV</td>
<td>Icing severity</td>
</tr>
<tr>
<td>ICTT</td>
<td>Icing Top</td>
</tr>
<tr>
<td>ICTT</td>
<td>Icing Top</td>
</tr>
<tr>
<td>CTP</td>
<td>In-Cloud Turbulence</td>
</tr>
<tr>
<td>IRBand4</td>
<td>Infrared Imagery</td>
</tr>
<tr>
<td>INSTRR</td>
<td>Instantaneous rain rate</td>
</tr>
<tr>
<td>LIPMF</td>
<td>Integrated column particulate matter (fine)</td>
</tr>
<tr>
<td>LIPMF</td>
<td>Integrated column particulate matter (fine)</td>
</tr>
<tr>
<td>ILIQW</td>
<td>Integrated Liquid Water</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ILW</td>
<td>Integrated liquid water</td>
</tr>
<tr>
<td>TSI</td>
<td>Integrated Solar Irradiance</td>
</tr>
<tr>
<td>INTFD</td>
<td>Interface Depths</td>
</tr>
<tr>
<td>IMFTSW</td>
<td>Inverse Mean Frequency of The Total Swell</td>
</tr>
<tr>
<td>IMFWW</td>
<td>Inverse Mean Frequency of The Wind Waves</td>
</tr>
<tr>
<td>IMWF</td>
<td>Inverse Mean Wave Frequency</td>
</tr>
<tr>
<td>IONDEN</td>
<td>Ion Density</td>
</tr>
<tr>
<td>IDRL</td>
<td>Ionospheric D-region level</td>
</tr>
<tr>
<td>IERL</td>
<td>Ionospheric E-region level</td>
</tr>
<tr>
<td>IF1RL</td>
<td>Ionospheric F1-region level</td>
</tr>
<tr>
<td>IF2RL</td>
<td>Ionospheric F2-region level</td>
</tr>
<tr>
<td>IONTMP</td>
<td>Ion Temperature</td>
</tr>
<tr>
<td>THEL</td>
<td>Isentropic (theta) level</td>
</tr>
<tr>
<td>ISBL</td>
<td>Isobaric Surface</td>
</tr>
<tr>
<td>TMPL</td>
<td>Isothermal Level</td>
</tr>
<tr>
<td>KX</td>
<td>K index</td>
</tr>
<tr>
<td>KENG</td>
<td>Kinetic Energy</td>
</tr>
<tr>
<td>MELBRNE</td>
<td>K NES 1</td>
</tr>
<tr>
<td>KOX</td>
<td>K0 index</td>
</tr>
<tr>
<td>BENINX</td>
<td>Kurtosis of The Sea Surface Elevation Due to Waves</td>
</tr>
<tr>
<td>LAND</td>
<td>Land cover (0=sea, 1=land)</td>
</tr>
<tr>
<td>LANDN</td>
<td>Land-sea coverage (nearest neighbor)</td>
</tr>
<tr>
<td>LSPA</td>
<td>Land Surface Precipitation Accumulation</td>
</tr>
<tr>
<td>LANDU</td>
<td>Land use</td>
</tr>
<tr>
<td>LAPR</td>
<td>Lapse rate</td>
</tr>
<tr>
<td>LRGHR</td>
<td>Large Scale Condensate Heating rate</td>
</tr>
<tr>
<td>LRGMR</td>
<td>Large scale moistening rate</td>
</tr>
<tr>
<td>NCPCP</td>
<td>Large-Scale Precipitation (non-convective)</td>
</tr>
<tr>
<td>NCPCP</td>
<td>Large scale precipitation (non-convective)</td>
</tr>
<tr>
<td>LSPrate</td>
<td>Large Scale Precipitation Rate</td>
</tr>
<tr>
<td>LSPRate</td>
<td>Large Scale Precipitation Rate</td>
</tr>
<tr>
<td>LSSRate</td>
<td>Large Scale Snowfall Rate</td>
</tr>
<tr>
<td>LSSRate</td>
<td>Large Scale Snowfall Rate</td>
</tr>
<tr>
<td>LSSRWE</td>
<td>Large Scale Snowfall Rate Water Equivalent</td>
</tr>
<tr>
<td>LSSRWE</td>
<td>Large Scale Snowfall Rate Water Equivalent</td>
</tr>
<tr>
<td>SNOL</td>
<td>Large-Scale Snow</td>
</tr>
<tr>
<td>LSWP</td>
<td>Large scale snow</td>
</tr>
<tr>
<td>LSWP</td>
<td>Large Scale Water Precipitation (Non-Convective)</td>
</tr>
<tr>
<td>LHTFL</td>
<td>Latent heat net flux</td>
</tr>
<tr>
<td>NLAT</td>
<td>Latitude (-90 to +90)</td>
</tr>
<tr>
<td>NLATN</td>
<td>Latitude (nearest neighbor) (-90 to +90)</td>
</tr>
<tr>
<td>LAPP</td>
<td>Latitude of Pressure Point</td>
</tr>
<tr>
<td>LAUV</td>
<td>Latitude of U Wind Component of Velocity</td>
</tr>
<tr>
<td>LAVV</td>
<td>Latitude of V Wind Component of Velocity</td>
</tr>
<tr>
<td>NONE</td>
<td>Layer Between Two Depths Below Ocean Surface</td>
</tr>
<tr>
<td>OLYR</td>
<td>Layer between two depths below ocean surface</td>
</tr>
<tr>
<td>OLYR</td>
<td>Layer Between Two Depths Below Ocean Surface</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBTHL</td>
<td>Layer Between Two Hybrid Levels</td>
</tr>
<tr>
<td>NONE</td>
<td>Layer Between Two Hybrid Levels</td>
</tr>
<tr>
<td>LMBSR</td>
<td>Layer-maximum base reflectivity</td>
</tr>
<tr>
<td>LOS</td>
<td>Layer Ocean Surface and 26C Ocean Isothermal Level</td>
</tr>
<tr>
<td>NONE</td>
<td>Layer Ocean Surface and 26C Ocean Isothermal Level</td>
</tr>
<tr>
<td>LAYTH</td>
<td>Layer Thickness</td>
</tr>
<tr>
<td>LAI</td>
<td>Leaf Area Index</td>
</tr>
<tr>
<td>PDLY</td>
<td>Level at Specified Pressure Difference from Ground to Level</td>
</tr>
<tr>
<td>SPDL</td>
<td>Level at Specified Pressure Difference from Ground to Level</td>
</tr>
<tr>
<td>0DEG</td>
<td>Level of 0°C Isotherm</td>
</tr>
<tr>
<td>ADCL</td>
<td>Level of Adiabatic Condensation Lifted from the Surface</td>
</tr>
<tr>
<td>CTL</td>
<td>Level of Cloud Tops</td>
</tr>
<tr>
<td>LTNG</td>
<td>Lightning</td>
</tr>
<tr>
<td>LTNG</td>
<td>Lightning</td>
</tr>
<tr>
<td>LMBINT</td>
<td>Limb Intensity</td>
</tr>
<tr>
<td>ARAIN</td>
<td>Liquid precipitation (rainfall)</td>
</tr>
<tr>
<td>ARAIN</td>
<td>Liquid precipitation (rainfall)</td>
</tr>
<tr>
<td>LSOIL</td>
<td>Liquid soil moisture content (non-frozen)</td>
</tr>
<tr>
<td>LIQVSM</td>
<td>Liquid Volumetric Soil Moisture (Non-Frozen)</td>
</tr>
<tr>
<td>SOILL</td>
<td>Liquid volumetric soil moisture (non-frozen)</td>
</tr>
<tr>
<td>LOPP</td>
<td>Longitude of Pressure Point</td>
</tr>
<tr>
<td>LOUV</td>
<td>Longitude of U Wind Component of Velocity</td>
</tr>
<tr>
<td>LOVV</td>
<td>Longitude of V Wind Component of Velocity</td>
</tr>
<tr>
<td>LWAVR</td>
<td>Long wave radiation flux</td>
</tr>
<tr>
<td>LWHR</td>
<td>Long-Wave Radiative Heating Rate</td>
</tr>
<tr>
<td>LCBL</td>
<td>Low cloud bottom level</td>
</tr>
<tr>
<td>LCBL</td>
<td>Low Cloud Bottom Level</td>
</tr>
<tr>
<td>LCLC</td>
<td>Low cloud cover</td>
</tr>
<tr>
<td>LCLY</td>
<td>Low Cloud Layer</td>
</tr>
<tr>
<td>LCY</td>
<td>Low cloud layer</td>
</tr>
<tr>
<td>LCY</td>
<td>Low Cloud Layer</td>
</tr>
<tr>
<td>LCTL</td>
<td>Low cloud top level</td>
</tr>
<tr>
<td>LCTL</td>
<td>Low Cloud Top Level</td>
</tr>
<tr>
<td>LLSM</td>
<td>Lower layer soil moisture</td>
</tr>
<tr>
<td>LSBLSW</td>
<td>Lowest Bottom Level of Supercooled Liquid Water Layer</td>
</tr>
<tr>
<td>LSCLW</td>
<td>Lowest bottom level of supercooled liquid water layer</td>
</tr>
<tr>
<td>LSCLW</td>
<td>Lowest Bottom Level of Supercooled Liquid Water Layer</td>
</tr>
<tr>
<td>LLTW</td>
<td>Lowest level of the wet bulb zero</td>
</tr>
<tr>
<td>LLTW</td>
<td>Lowest Level of the Wet Bulb Zero</td>
</tr>
<tr>
<td>LWBZ</td>
<td>Lowest Level of the Wet Bulb Zero</td>
</tr>
<tr>
<td>WBZ</td>
<td>Lowest Level of the Wet Bulb Zero</td>
</tr>
<tr>
<td>LowLayerCompositeReflectivity</td>
<td>Low Layer Composite Reflectivity (0-24 kft)</td>
</tr>
<tr>
<td>LAVNI</td>
<td>Low-Level aviation interest</td>
</tr>
<tr>
<td>MergedAzShear02kmAGL</td>
<td>Low-Level Azimuthal Shear (0-2km AGL)</td>
</tr>
<tr>
<td>LLCompositeReflectivity</td>
<td>Low-Level Composite Reflectivity</td>
</tr>
<tr>
<td>HeightLLCompositeReflectivity</td>
<td>Low-Level Composite Reflectivity Height</td>
</tr>
<tr>
<td>RotationTrackLL120min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (120 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackLL1440min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (1440 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackLL240min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (240 min. accum.)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>RotationTrackLL30min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (30 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackLL360min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (360 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackLL60min</td>
<td>Low-Level Rotation Tracks 0-2km AGL (60 min. accum.)</td>
</tr>
<tr>
<td>BTOT</td>
<td>Magnetic Field Magnitude</td>
</tr>
<tr>
<td>MTHA</td>
<td>Main thermocline anomaly</td>
</tr>
<tr>
<td>MTHD</td>
<td>Main thermocline depth</td>
</tr>
<tr>
<td>LMH</td>
<td>Mass Point Model Surface</td>
</tr>
<tr>
<td>MAXAH</td>
<td>Maximum absolute humidity</td>
</tr>
<tr>
<td>MAXAH</td>
<td>Maximum Absolute Humidity</td>
</tr>
<tr>
<td>MACAT</td>
<td>Maximum Cloud Air Turbulence Potential</td>
</tr>
<tr>
<td>REFC</td>
<td>Maximum/Composite radar reflectivity</td>
</tr>
<tr>
<td>MTHIE</td>
<td>Maximum equivalent potential temperature level</td>
</tr>
<tr>
<td>MTHIE</td>
<td>Maximum Equivalent Potential Temperature level</td>
</tr>
<tr>
<td>MESH</td>
<td>Maximum Estimated Size of Hail (MESH)</td>
</tr>
<tr>
<td>MAIP</td>
<td>Maximum Icing Potential</td>
</tr>
<tr>
<td>MAXWH</td>
<td>Maximum Individual Wave Height</td>
</tr>
<tr>
<td>PRPMax</td>
<td>Maximum Precipitation Return Period</td>
</tr>
<tr>
<td>QPEFFGMax</td>
<td>Maximum QPE-to-FFG Ratio</td>
</tr>
<tr>
<td>MAXRH</td>
<td>Maximum relative humidity</td>
</tr>
<tr>
<td>MAXRH</td>
<td>Maximum Relative Humidity</td>
</tr>
<tr>
<td>MXSALB</td>
<td>Maximum snow albedo</td>
</tr>
<tr>
<td>MXSALB</td>
<td>Maximum Snow Albedo*</td>
</tr>
<tr>
<td>QMAX</td>
<td>Maximum specific humidity at 2m</td>
</tr>
<tr>
<td>TMAX</td>
<td>Maximum temperature</td>
</tr>
<tr>
<td>MWSL</td>
<td>Maximum Wind Level</td>
</tr>
<tr>
<td>MAXWS</td>
<td>Maximum wind speed</td>
</tr>
<tr>
<td>MACTP</td>
<td>Max in-Cloud Turbulence Potential</td>
</tr>
<tr>
<td>MECAT</td>
<td>Mean Cloud Air Turbulence Potential</td>
</tr>
<tr>
<td>MEI</td>
<td>Mean Icing Potential</td>
</tr>
<tr>
<td>MECTP</td>
<td>Mean in-Cloud Turbulence Potential</td>
</tr>
<tr>
<td>MWSPER</td>
<td>Mean period of combined wind waves and swell</td>
</tr>
<tr>
<td>SWPER</td>
<td>Mean period of swell waves</td>
</tr>
<tr>
<td>WVPER</td>
<td>Mean period of wind waves</td>
</tr>
<tr>
<td>MSL</td>
<td>Mean Sea Level</td>
</tr>
<tr>
<td>M2SPW</td>
<td>Mean square slope of waves</td>
</tr>
<tr>
<td>M2PTSW</td>
<td>Mean Zero-Crossing Period of The Total Swell</td>
</tr>
<tr>
<td>M2PW</td>
<td>Mean Zero-Crossing Period of The Wind Waves</td>
</tr>
<tr>
<td>MZWPER</td>
<td>Mean Zero-Crossing Wave Period</td>
</tr>
<tr>
<td>MDC</td>
<td>Medium cloud cover</td>
</tr>
<tr>
<td>MergedBaseReflectivityQC</td>
<td>Merged Base Reflectivity</td>
</tr>
<tr>
<td>MergedReflectivityAtLowestAltitude</td>
<td>Merged Reflectivity At Lowest Altitude (RALA)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MESHTrack60min</td>
<td>MESH Tracks (60 min. accum.)</td>
</tr>
<tr>
<td>MCBL</td>
<td>Middle cloud bottom level</td>
</tr>
<tr>
<td>MCBL</td>
<td>Middle Cloud Bottom Level</td>
</tr>
<tr>
<td>MCLY</td>
<td>Middle Cloud Layer</td>
</tr>
<tr>
<td>MCY</td>
<td>Middle cloud layer</td>
</tr>
<tr>
<td>MCY</td>
<td>Middle Cloud Layer</td>
</tr>
<tr>
<td>MCTL</td>
<td>Middle cloud top level</td>
</tr>
<tr>
<td>MCTL</td>
<td>Middle Cloud Top Level</td>
</tr>
<tr>
<td>MergedAzShear36kmAGL</td>
<td>Mid-Level Azimuthal Shear (3-6km AGL)</td>
</tr>
<tr>
<td>RotationTrackML120min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (120 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackML1440min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (1440 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackML240min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (240 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackML30min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (30 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackML360min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (360 min. accum.)</td>
</tr>
<tr>
<td>RotationTrackML60min</td>
<td>Mid-Level Rotation Tracks 3-6km AGL (60 min. accum.)</td>
</tr>
<tr>
<td>RSMIN</td>
<td>Minimal stomatal resistance</td>
</tr>
<tr>
<td>DEPMN</td>
<td>Minimum dew point depression</td>
</tr>
<tr>
<td>MINRH</td>
<td>Minimum Relative Humidity</td>
</tr>
<tr>
<td>QMIN</td>
<td>Minimum specific humidity at 2m</td>
</tr>
<tr>
<td>TMIN</td>
<td>Minimum temperature</td>
</tr>
<tr>
<td>Entire Atmosphere</td>
<td>Missing200</td>
</tr>
<tr>
<td>MIXHT</td>
<td>mixed layer depth</td>
</tr>
<tr>
<td>MIXHT</td>
<td>Mixed Layer Depth</td>
</tr>
<tr>
<td>MIXL</td>
<td>Mixed Layer Depth</td>
</tr>
<tr>
<td>MLYNO</td>
<td>Model Layer number (From bottom up)</td>
</tr>
<tr>
<td>MTHT</td>
<td>Model terrain height</td>
</tr>
<tr>
<td>MRCONO</td>
<td>Moderate risk convective outlook</td>
</tr>
<tr>
<td>MSTAV</td>
<td>Moisture availability</td>
</tr>
<tr>
<td>UFLX</td>
<td>Momentum flux, u component</td>
</tr>
<tr>
<td>VFLX</td>
<td>Momentum flux, v component</td>
</tr>
<tr>
<td>MNTSF</td>
<td>Montgomery stream function</td>
</tr>
<tr>
<td>MSLET</td>
<td>MSLP (Eta model reduction)</td>
</tr>
<tr>
<td>MSLPM</td>
<td>MSLP (MAPS System Reduction)</td>
</tr>
<tr>
<td>NLSGP</td>
<td>Natural Log of Surface Pressure</td>
</tr>
<tr>
<td>NBDSF</td>
<td>Near IR Beam Downward Solar Flux</td>
</tr>
<tr>
<td>NBSALB</td>
<td>Near IR, Black Sky Albedo</td>
</tr>
<tr>
<td>NDDSF</td>
<td>Near IR Diffuse Downward Solar Flux</td>
</tr>
<tr>
<td>NWSALB</td>
<td>Near IR, White Sky Albedo</td>
</tr>
<tr>
<td>AOHFLX</td>
<td>Net Air-Ocean Heat Flux</td>
</tr>
<tr>
<td>NLWRCSC</td>
<td>Net Long-Wave Radiation Flux, Clear Sky</td>
</tr>
<tr>
<td>NLWRS</td>
<td>Net long wave radiation flux (surface)</td>
</tr>
<tr>
<td>NLWRT</td>
<td>Net long wave radiation flux (top of atmosphere)</td>
</tr>
<tr>
<td>NLWRF</td>
<td>Net Long-Wave Radiation Flux</td>
</tr>
<tr>
<td>NSWRFCS</td>
<td>Net Short-Wave Radiation Flux, Clear Sky</td>
</tr>
<tr>
<td>NSWRS</td>
<td>Net short-wave radiation flux (surface)</td>
</tr>
<tr>
<td>NSWRT</td>
<td>Net short-wave radiation flux (top of atmosphere)</td>
</tr>
<tr>
<td>NSWRF</td>
<td>Net Short Wave Radiation Flux</td>
</tr>
<tr>
<td>NTAT</td>
<td>Nominal Top of the Atmosphere</td>
</tr>
<tr>
<td>CDLYR</td>
<td>Non-convective cloud cover</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CDLYR</td>
<td>Non-Convective Cloud Cover non-dim</td>
</tr>
<tr>
<td>NWSTR</td>
<td>Normalised Waves Stress</td>
</tr>
<tr>
<td>NDVI</td>
<td>Normalized Difference Vegetation Index</td>
</tr>
<tr>
<td>NCIP</td>
<td>Number concentration for ice particles</td>
</tr>
<tr>
<td>MIXLY</td>
<td>Number of mixed layers next to surface</td>
</tr>
<tr>
<td>NPIXU</td>
<td>Number Of Pixels Used</td>
</tr>
<tr>
<td>RLYRS</td>
<td>Number of soil layers in root zone</td>
</tr>
<tr>
<td>RLYRS</td>
<td>Number of soil layers in root zone</td>
</tr>
<tr>
<td>RLYRS</td>
<td>Number of soil layers in root zone</td>
</tr>
<tr>
<td>OHC</td>
<td>Ocean Heat Content</td>
</tr>
<tr>
<td>OITL</td>
<td>Ocean Isotherm Level (1/10 deg C)</td>
</tr>
<tr>
<td>NONE</td>
<td>Ocean Isotherm Level</td>
</tr>
<tr>
<td>OML</td>
<td>Ocean Mixed Layer</td>
</tr>
<tr>
<td>P2OMLT</td>
<td>Ocean Mixed Layer Potential Density (Reference 2000m)</td>
</tr>
<tr>
<td>OMLU</td>
<td>Ocean Mixed Layer U Velocity</td>
</tr>
<tr>
<td>OMLV</td>
<td>Ocean Mixed Layer V Velocity</td>
</tr>
<tr>
<td>ELEV</td>
<td>Ocean Surface Elevation Relative to Geoid</td>
</tr>
<tr>
<td>OMGALF</td>
<td>Omega (Dp/Dt) divide by density</td>
</tr>
<tr>
<td>EWATR</td>
<td>Open water evaporation (standing water)</td>
</tr>
<tr>
<td>OSD</td>
<td>Ordered Sequence of Data</td>
</tr>
<tr>
<td>OSEQ</td>
<td>Ordered Sequence of Data</td>
</tr>
<tr>
<td>OZCON</td>
<td>Ozone Concentration (PPB)</td>
</tr>
<tr>
<td>OZMAX1</td>
<td>Ozone Daily Max from 1-hour Average</td>
</tr>
<tr>
<td>OZMAX8</td>
<td>Ozone Daily Max from 8-hour Average</td>
</tr>
<tr>
<td>O3MR</td>
<td>Ozone Mixing Ratio</td>
</tr>
<tr>
<td>O3MR</td>
<td>Ozone mixing ratio</td>
</tr>
<tr>
<td>O3MR</td>
<td>Ozone Mixing Ratio</td>
</tr>
<tr>
<td>POZO</td>
<td>Ozone production from col ozone term</td>
</tr>
<tr>
<td>POZT</td>
<td>Ozone production from temperature term</td>
</tr>
<tr>
<td>TOZ</td>
<td>Ozone tendency</td>
</tr>
<tr>
<td>VDFOZ</td>
<td>Ozone vertical diffusion</td>
</tr>
<tr>
<td>SIGPAR</td>
<td>Parallel Conductivity</td>
</tr>
<tr>
<td>PRATMP</td>
<td>Parallel Temperature</td>
</tr>
<tr>
<td>PLI</td>
<td>Parcel lifted index (to 500 mb)</td>
</tr>
<tr>
<td>PLSMDEN</td>
<td>Particle Number Density</td>
</tr>
<tr>
<td>PMTC</td>
<td>Particulate matter (coarse)</td>
</tr>
<tr>
<td>PMTC</td>
<td>Particulate matter (coarse)</td>
</tr>
<tr>
<td>LPMTF</td>
<td>Particulate matter (fine)</td>
</tr>
<tr>
<td>LPMTF</td>
<td>Particulate matter (fine)</td>
</tr>
<tr>
<td>PMTF</td>
<td>Particulate matter (fine)</td>
</tr>
<tr>
<td>PMTF</td>
<td>Particulate matter (fine)</td>
</tr>
<tr>
<td>PPERTS</td>
<td>Peak Period of The Total Swell</td>
</tr>
<tr>
<td>PPERWW</td>
<td>Peak Period of The Wind Waves</td>
</tr>
<tr>
<td>PWPER</td>
<td>Peak Wave Period</td>
</tr>
<tr>
<td>SIGPED</td>
<td>Pedersen Conductivity</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPFFP</td>
<td>Percent frozen precipitation</td>
</tr>
<tr>
<td>PCPP1</td>
<td>Percent pcpn in 1st 6-h sub-period of 24 hr period</td>
</tr>
<tr>
<td>PCPP2</td>
<td>Percent pcpn in 2nd 6-h sub-period of 24 hr period</td>
</tr>
<tr>
<td>PCPP3</td>
<td>Percent pcpn in 3rd 6-h sub-period of 24 hr period</td>
</tr>
<tr>
<td>PCPP4</td>
<td>Percent pcpn in 4th 6-h sub-period of 24 hr period</td>
</tr>
<tr>
<td>PCPSUB</td>
<td>Percent precipitation in a sub-period of an overall period (Encoded as per cent accumulation over the sub-period)</td>
</tr>
<tr>
<td>PMAXWH</td>
<td>Period of Maximum Individual Wave Height</td>
</tr>
<tr>
<td>PRITMP</td>
<td>Perpendicular Temperature</td>
</tr>
<tr>
<td>PHOTAR</td>
<td>Photosynthetically Active Radiation</td>
</tr>
<tr>
<td>PIXST</td>
<td>Pixel scene type</td>
</tr>
<tr>
<td>BLD</td>
<td>Planetary Boundary Layer</td>
</tr>
<tr>
<td>HPBL</td>
<td>Planetary boundary layer height</td>
</tr>
<tr>
<td>HPBLL</td>
<td>Planetary Boundary Layer Height</td>
</tr>
<tr>
<td>PLBL</td>
<td>Planetary Boundary Layer</td>
</tr>
<tr>
<td>PBLR</td>
<td>Planetary boundary layer regime</td>
</tr>
<tr>
<td>PBLRREG</td>
<td>Planetary Boundary Layer Regime</td>
</tr>
<tr>
<td>CNWAT</td>
<td>Plant canopy surface water</td>
</tr>
<tr>
<td>PDMAX1</td>
<td>PM 2.5 Daily Max from 1-hour Average</td>
</tr>
<tr>
<td>PDMAX24</td>
<td>PM 2.5 Daily Max from 24-hour Average</td>
</tr>
<tr>
<td>PEVAP</td>
<td>Potential Evaporation</td>
</tr>
<tr>
<td>PEVAP</td>
<td>Potential evaporation</td>
</tr>
<tr>
<td>PEVAP</td>
<td>Potential Evaporation</td>
</tr>
<tr>
<td>PEVPR</td>
<td>Potential Evaporation Rage</td>
</tr>
<tr>
<td>PEVPR</td>
<td>Potential evaporation rate</td>
</tr>
<tr>
<td>PEVPR</td>
<td>Potential Evaporation Rate</td>
</tr>
<tr>
<td>THZ0</td>
<td>Potential temperature at top of viscous sublayer</td>
</tr>
<tr>
<td>POT</td>
<td>Potential temperature</td>
</tr>
<tr>
<td>POT</td>
<td>Potential temperature (theta)</td>
</tr>
<tr>
<td>PV</td>
<td>Potential vorticity</td>
</tr>
<tr>
<td>PV</td>
<td>Potential vorticity</td>
</tr>
<tr>
<td>PVL</td>
<td>Potential Vorticity</td>
</tr>
<tr>
<td>PVORT</td>
<td>Potential vorticity</td>
</tr>
<tr>
<td>PVMWW</td>
<td>Potential Vorticity (Mass-Weighted)</td>
</tr>
<tr>
<td>PW</td>
<td>Precipitable water category</td>
</tr>
<tr>
<td>PWAT</td>
<td>Precipitable Water</td>
</tr>
<tr>
<td>PWAT</td>
<td>Precipitable water</td>
</tr>
<tr>
<td>PRCP</td>
<td>Precipitation</td>
</tr>
<tr>
<td>PRATE</td>
<td>Precipitation Rate</td>
</tr>
<tr>
<td>PRATE</td>
<td>Precipitation rate</td>
</tr>
<tr>
<td>PTYPE</td>
<td>Precipitation type</td>
</tr>
<tr>
<td>PTYPE</td>
<td>Precipitation Type</td>
</tr>
<tr>
<td>PR</td>
<td>Precip rate</td>
</tr>
<tr>
<td>(See note 3)</td>
<td>Predominant Weather</td>
</tr>
<tr>
<td>PWATHER</td>
<td>Predominant Weather</td>
</tr>
<tr>
<td>PALT</td>
<td>Pressure altitude</td>
</tr>
<tr>
<td>PRESA</td>
<td>Pressure anomaly</td>
</tr>
<tr>
<td>PCBB</td>
<td>Pressure at Cumulonimbus Bas</td>
</tr>
<tr>
<td>PCBT</td>
<td>Pressure at Cumulonimbus To</td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>PECBB</td>
<td>Pressure at Embedded Cumulonimbus Bas</td>
<td>Pa</td>
</tr>
<tr>
<td>PECBT</td>
<td>Pressure at Embedded Cumulonimbus To</td>
<td>Pa</td>
</tr>
<tr>
<td>PRESDEV</td>
<td>Pressure deviation from ground to level</td>
<td>Pa</td>
</tr>
<tr>
<td>PRESD</td>
<td>Pressure deviation from mean sea level</td>
<td>Pa</td>
</tr>
<tr>
<td>PRESN</td>
<td>Pressure (nearest grid point)</td>
<td>Pa</td>
</tr>
<tr>
<td>PLPL</td>
<td>Pressure of level from which parcel was lifted</td>
<td>Pa</td>
</tr>
<tr>
<td>PLPL</td>
<td>Pressure of most parcel with highest theta-e in lowest 300 mb</td>
<td>Pa</td>
</tr>
<tr>
<td>P</td>
<td>Pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>PRES</td>
<td>Pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>PRMSL</td>
<td>Pressure reduced to MSL</td>
<td>Pa</td>
</tr>
<tr>
<td>PTEND</td>
<td>Pressure tendency</td>
<td>Pa</td>
</tr>
<tr>
<td>DIRPW</td>
<td>Primary wave direction</td>
<td>Degree</td>
</tr>
<tr>
<td>PERPW</td>
<td>Primary wave mean period</td>
<td>s</td>
</tr>
<tr>
<td>POP</td>
<td>Probability of 0.01 inches of precipitation</td>
<td>%</td>
</tr>
<tr>
<td>POP</td>
<td>Probability of 0.01 inch of precipitation (POP)</td>
<td>%</td>
</tr>
<tr>
<td>PROCON</td>
<td>Probability of Convection</td>
<td>%</td>
</tr>
<tr>
<td>PFFG</td>
<td>Probability of Excessive Rain</td>
<td>%</td>
</tr>
<tr>
<td>CPOZP</td>
<td>Probability of Freezing Precipitation</td>
<td>%</td>
</tr>
<tr>
<td>PFREZPREC</td>
<td>Probability of Freezing Precipitation</td>
<td>%</td>
</tr>
<tr>
<td>CPOFP</td>
<td>Probability of Frozen Precipitation</td>
<td>%</td>
</tr>
<tr>
<td>PFROZPREC</td>
<td>Probability of Frozen Precipitation</td>
<td>%</td>
</tr>
<tr>
<td>PPFFG</td>
<td>Probability of precipitation exceeding flash flood guidance values</td>
<td>%</td>
</tr>
<tr>
<td>POSH</td>
<td>Probability of Severe Hail (POSH)</td>
<td>%</td>
</tr>
<tr>
<td>WarmRainProb</td>
<td>Probability of Warm Rain</td>
<td>%</td>
</tr>
<tr>
<td>CWR</td>
<td>Probability of Wetting Rain, exceeding in 0.10” in a given time period</td>
<td>%</td>
</tr>
<tr>
<td>PTAN</td>
<td>Prob of Temperature above normal</td>
<td>%</td>
</tr>
<tr>
<td>PTBN</td>
<td>Prob of Temperature below normal</td>
<td>%</td>
</tr>
<tr>
<td>PTNN</td>
<td>Prob of Temperature near normal</td>
<td>%</td>
</tr>
<tr>
<td>PPAN</td>
<td>Prob of Total Precipitation above normal</td>
<td>%</td>
</tr>
<tr>
<td>PPBN</td>
<td>Prob of Total Precipitation below normal</td>
<td>%</td>
</tr>
<tr>
<td>PPNN</td>
<td>Prob of Total Precipitation near normal</td>
<td>%</td>
</tr>
<tr>
<td>PROTDEN</td>
<td>Proton Density</td>
<td>m$^3$</td>
</tr>
<tr>
<td>DIFPFLUX</td>
<td>Proton Flux (Differential)</td>
<td>(m$^2$ s sr eV)$^1$</td>
</tr>
<tr>
<td>INTPFLUX</td>
<td>Proton Flux (Integral)</td>
<td>(m$^2$ s sr)$^1$</td>
</tr>
<tr>
<td>PROTTMP</td>
<td>Proton Temperature</td>
<td>K</td>
</tr>
<tr>
<td>EPOT</td>
<td>Pseudo-adiabatic potential temperature or equivalent potential temperature</td>
<td>K</td>
</tr>
<tr>
<td>MountainMapperQPE12H</td>
<td>QPE - Mountain Mapper (12 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE01H</td>
<td>QPE - Mountain Mapper (1 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE24H</td>
<td>QPE - Mountain Mapper (24 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE03H</td>
<td>QPE - Mountain Mapper (3 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE48H</td>
<td>QPE - Mountain Mapper (48 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE06H</td>
<td>QPE - Mountain Mapper (6 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>MountainMapperQPE72H</td>
<td>QPE - Mountain Mapper (72 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>GaugeOnlyQPE12H</td>
<td>QPE - Radar Gauge Only (12 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>GaugeOnlyQPE01H</td>
<td>QPE - Radar Gauge Only (1 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>GaugeOnlyQPE24H</td>
<td>QPE - Radar Gauge Only (24 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>GaugeOnlyQPE03H</td>
<td>QPE - Radar Gauge Only (3 hr. accum.)</td>
<td>mm</td>
</tr>
<tr>
<td>GaugeOnlyQPE48H</td>
<td>QPE - Radar Gauge Only (48 hr. accum.)</td>
<td>mm</td>
</tr>
</tbody>
</table>

3.1. **Quick Example**
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaugeOnlyQPE06H</td>
<td>QPE - Radar Gauge Only (6 hr. accum.)</td>
</tr>
<tr>
<td>GaugeOnlyQPE72H</td>
<td>QPE - Radar Gauge Only (72 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE12H</td>
<td>QPE - Radar Only (12 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE01H</td>
<td>QPE - Radar Only (1 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE24H</td>
<td>QPE - Radar Only (24 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE03H</td>
<td>QPE - Radar Only (3 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE48H</td>
<td>QPE - Radar Only (48 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE06H</td>
<td>QPE - Radar Only (6 hr. accum.)</td>
</tr>
<tr>
<td>RadarOnlyQPE72H</td>
<td>QPE - Radar Only (72 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE12H</td>
<td>QPE - Radar with Gauge Bias Correction (12 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE01H</td>
<td>QPE - Radar with Gauge Bias Correction (1 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE24H</td>
<td>QPE - Radar with Gauge Bias Correction (24 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE03H</td>
<td>QPE - Radar with Gauge Bias Correction (3 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE48H</td>
<td>QPE - Radar with Gauge Bias Correction (48 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE06H</td>
<td>QPE - Radar with Gauge Bias Correction (6 hr. accum.)</td>
</tr>
<tr>
<td>GaugeCorrQPE72H</td>
<td>QPE - Radar with Gauge Bias Correction (72 hr. accum.)</td>
</tr>
<tr>
<td>PrecipRate</td>
<td>Radar Precipitation Rate (SPR)</td>
</tr>
<tr>
<td>RadarQualityIndex</td>
<td>Radar Quality Index (RQI)</td>
</tr>
<tr>
<td>RDSP1</td>
<td>Radar spectra (1)</td>
</tr>
<tr>
<td>RDSP2</td>
<td>Radar spectra (2)</td>
</tr>
<tr>
<td>RDSP3</td>
<td>Radar spectra (3)</td>
</tr>
<tr>
<td>RDNUML</td>
<td>Radial number (2pi/lambda)</td>
</tr>
<tr>
<td>SWRAD</td>
<td>Radiance (with respect to wave length)</td>
</tr>
<tr>
<td>LWRAD</td>
<td>Radiance (with respect to wave number)</td>
</tr>
<tr>
<td>EPSR</td>
<td>Radiative emissivity</td>
</tr>
<tr>
<td>EPSR</td>
<td>Radiative emissivity</td>
</tr>
<tr>
<td>FRAIN</td>
<td>Rain fraction of total cloud water</td>
</tr>
<tr>
<td>FRAIN</td>
<td>Rain Fraction of Total Liquid Water</td>
</tr>
<tr>
<td>FRAINL</td>
<td>Rain Fraction of Total Liquid Water</td>
</tr>
<tr>
<td>RWMR</td>
<td>Rain Mixing Ratio</td>
</tr>
<tr>
<td>RWMR</td>
<td>Rain mixing ratio</td>
</tr>
<tr>
<td>RPRATE</td>
<td>Rain Precipitation Rate</td>
</tr>
<tr>
<td>RPRATE</td>
<td>Rain Precipitation Rate</td>
</tr>
<tr>
<td>RDRIP</td>
<td>Rate of water dropping from canopy to ground</td>
</tr>
<tr>
<td>Rift1</td>
<td>Rate of water dropping from canopy to ground</td>
</tr>
<tr>
<td>MergedReflectivityComposite</td>
<td>Raw Composite Reflectivity Mosaic</td>
</tr>
<tr>
<td>MergedReflectivity</td>
<td>Raw Merged Base Reflectivity</td>
</tr>
<tr>
<td>RFL06</td>
<td>Reflectance in 0.6 Micron Channel</td>
</tr>
<tr>
<td>RFL08</td>
<td>Reflectance in 0.8 Micron Channel</td>
</tr>
<tr>
<td>RFL16</td>
<td>Reflectance in 1.6 Micron Channel</td>
</tr>
<tr>
<td>RFL39</td>
<td>Reflectance in 3.9 Micron Channel</td>
</tr>
<tr>
<td>Reflectivity0C</td>
<td>Reflectivity at 0C</td>
</tr>
<tr>
<td>ReflectivityM10C</td>
<td>Reflectivity at -10C</td>
</tr>
<tr>
<td>ReflectivityM15C</td>
<td>Reflectivity at -15C</td>
</tr>
<tr>
<td>ReflectivityM20C</td>
<td>Reflectivity at -20C</td>
</tr>
<tr>
<td>ReflectivityM25C</td>
<td>Reflectivity at -25C</td>
</tr>
<tr>
<td>ReflectivityAtLowestAltitude</td>
<td>Reflectivity At Lowest Altitude (RALA)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>REFD</td>
<td>Reflectivity</td>
</tr>
<tr>
<td>RAZA</td>
<td>Relative Azimuth Angle</td>
</tr>
<tr>
<td>RELD</td>
<td>Relative divergence</td>
</tr>
<tr>
<td>REV</td>
<td>Relative Error Variance</td>
</tr>
<tr>
<td>RH</td>
<td>Relative humidity</td>
</tr>
<tr>
<td>R H</td>
<td>Relative Humidity</td>
</tr>
<tr>
<td>RHPW</td>
<td>Relative Humidity with Respect to Precipitable Water</td>
</tr>
<tr>
<td>RELV</td>
<td>Relative vorticity</td>
</tr>
<tr>
<td>RSSC</td>
<td>Remotely sensed snow cover</td>
</tr>
<tr>
<td>RI</td>
<td>Richardson number</td>
</tr>
<tr>
<td>FRIME</td>
<td>Rime Factor</td>
</tr>
<tr>
<td>RIME</td>
<td>Rime Factor</td>
</tr>
<tr>
<td>RIME</td>
<td>Rime factor</td>
</tr>
<tr>
<td>RIME</td>
<td>Rime Factor</td>
</tr>
<tr>
<td>SFCRH</td>
<td>Roughness length for heat</td>
</tr>
<tr>
<td>SALTY</td>
<td>Salinity</td>
</tr>
<tr>
<td>SLTFL</td>
<td>Salt Flux</td>
</tr>
<tr>
<td>SATD</td>
<td>Saturation deficit</td>
</tr>
<tr>
<td>SAT D</td>
<td>Saturation Deficit</td>
</tr>
<tr>
<td>SATOSM</td>
<td>Saturation Of Soil Moisture</td>
</tr>
<tr>
<td>SCALB</td>
<td>Scaled albedo</td>
</tr>
<tr>
<td>SCBT</td>
<td>Scaled brightness temperature</td>
</tr>
<tr>
<td>SCCTP</td>
<td>Scaled cloud top pressure</td>
</tr>
<tr>
<td>SCLI</td>
<td>Scaled lifted index</td>
</tr>
<tr>
<td>SCPW</td>
<td>Scaled precipitable water</td>
</tr>
<tr>
<td>SCRAD</td>
<td>Scaled radiance</td>
</tr>
<tr>
<td>SCST</td>
<td>Scaled skin temperature</td>
</tr>
<tr>
<td>SCESTUWIND</td>
<td>Scatterometer Estimated U Wind Component</td>
</tr>
<tr>
<td>SCESTVWIND</td>
<td>Scatterometer Estimated V Wind Component</td>
</tr>
<tr>
<td>SCINT</td>
<td>Scintillation</td>
</tr>
<tr>
<td>SEAB</td>
<td>Sea Bottom</td>
</tr>
<tr>
<td>SeamlessHSR</td>
<td>Seamless Hybrid Scan Reflectivity (SHSR)</td>
</tr>
<tr>
<td>SeamlessHSRHeight</td>
<td>Seamless Hybrid Scan Reflectivity (SHSR) Height</td>
</tr>
<tr>
<td>SSHG</td>
<td>Sea Surface Height Relative to Geoid</td>
</tr>
<tr>
<td>DIRSW</td>
<td>Secondary wave direction</td>
</tr>
<tr>
<td>PERSW</td>
<td>Secondary wave mean periods</td>
</tr>
<tr>
<td>s</td>
<td>Seconds prior to initial reference time (defined in Section 1)</td>
</tr>
<tr>
<td>TSEC</td>
<td>Seconds prior to initial reference time</td>
</tr>
<tr>
<td>TSEC</td>
<td>Seconds Prior To Initial Reference Time</td>
</tr>
<tr>
<td>SHTFL</td>
<td>Sensible heat net flux</td>
</tr>
<tr>
<td>SHI</td>
<td>Severe Hail Index (SHI)</td>
</tr>
<tr>
<td>SCBL</td>
<td>Shallow convective cloud bottom level</td>
</tr>
<tr>
<td>SCCBT</td>
<td>Shallow Convective Cloud Bottom Level</td>
</tr>
<tr>
<td>SCTL</td>
<td>Shallow Convective Cloud Top Level</td>
</tr>
<tr>
<td>SCTL</td>
<td>Shallow convective cloud top level</td>
</tr>
<tr>
<td>SCTL</td>
<td>Shallow Convective Cloud Top Level</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHAHR</td>
<td>Shallow Convective Heating rate</td>
</tr>
<tr>
<td>SHAMR</td>
<td>Shallow Convective Moistening Rate</td>
</tr>
<tr>
<td>SHAMR</td>
<td>Shallow Convective Moistening Rate</td>
</tr>
<tr>
<td>SWAVR</td>
<td>Short wave radiation flux</td>
</tr>
<tr>
<td>SGCVV</td>
<td>Sigma coordinate vertical velocity</td>
</tr>
<tr>
<td>SIGL</td>
<td>Sigma Level</td>
</tr>
<tr>
<td>SHAILPRO</td>
<td>Significant Hail probability</td>
</tr>
<tr>
<td>SIGHAILPROB</td>
<td>Significant Hail probability</td>
</tr>
<tr>
<td>HTSGW</td>
<td>Significant height of combined wind waves and swell</td>
</tr>
<tr>
<td>SWELL</td>
<td>Significant height of swell waves</td>
</tr>
<tr>
<td>WVHGT</td>
<td>Significant height of wind waves (m)</td>
</tr>
<tr>
<td>SIGTRNDPROB</td>
<td>Significant Tornado probability</td>
</tr>
<tr>
<td>STORPROB</td>
<td>Significant Tornado probability</td>
</tr>
<tr>
<td>SIGWINDPROB</td>
<td>Significant Wind probability</td>
</tr>
<tr>
<td>SBC123</td>
<td>Simulated Brightness Counts for GOES 12, Channel 3</td>
</tr>
<tr>
<td>SBC124</td>
<td>Simulated Brightness Counts for GOES 12, Channel 4</td>
</tr>
<tr>
<td>SBTA1610</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-10</td>
</tr>
<tr>
<td>SBTA1611</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-11</td>
</tr>
<tr>
<td>SBTA1612</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-12</td>
</tr>
<tr>
<td>SBTA1613</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-13</td>
</tr>
<tr>
<td>SBTA1614</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-14</td>
</tr>
<tr>
<td>SBTA1615</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-15</td>
</tr>
<tr>
<td>SBTA1616</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-16</td>
</tr>
<tr>
<td>SBTA1617</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-7</td>
</tr>
<tr>
<td>SBTA1618</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-8</td>
</tr>
<tr>
<td>SBTA1619</td>
<td>Simulated Brightness Temperature for ABI GOES-16, Band-9</td>
</tr>
<tr>
<td>SBTA1710</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-10</td>
</tr>
<tr>
<td>SBTA1711</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-11</td>
</tr>
<tr>
<td>SBTA1712</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-12</td>
</tr>
<tr>
<td>SBTA1713</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-13</td>
</tr>
<tr>
<td>SBTA1714</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-14</td>
</tr>
<tr>
<td>SBTA1715</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-15</td>
</tr>
<tr>
<td>SBTA1716</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-16</td>
</tr>
<tr>
<td>SBTA1717</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-7</td>
</tr>
<tr>
<td>SBTA1718</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-8</td>
</tr>
<tr>
<td>SBTA1719</td>
<td>Simulated Brightness Temperature for ABI GOES-17, Band-9</td>
</tr>
<tr>
<td>AMSRE10</td>
<td>Simulated Brightness Temperature for AMSRE on Aqua, Channel 10</td>
</tr>
<tr>
<td>AMSRE11</td>
<td>Simulated Brightness Temperature for AMSRE on Aqua, Channel 11</td>
</tr>
<tr>
<td>AMSRE12</td>
<td>Simulated Brightness Temperature for AMSRE on Aqua, Channel 12</td>
</tr>
<tr>
<td>AMSRE9</td>
<td>Simulated Brightness Temperature for AMSRE on Aqua, Channel 9</td>
</tr>
<tr>
<td>SBT112</td>
<td>Simulated Brightness Temperature for GOES 11, Channel 2</td>
</tr>
<tr>
<td>SBT113</td>
<td>Simulated Brightness Temperature for GOES 11, Channel 3</td>
</tr>
<tr>
<td>SBT114</td>
<td>Simulated Brightness Temperature for GOES 11, Channel 4</td>
</tr>
<tr>
<td>SBT115</td>
<td>Simulated Brightness Temperature for GOES 11, Channel 5</td>
</tr>
<tr>
<td>SBT122</td>
<td>Simulated Brightness Temperature for GOES 12, Channel 2</td>
</tr>
<tr>
<td>SBT123</td>
<td>Simulated Brightness Temperature for GOES 12, Channel 3</td>
</tr>
<tr>
<td>SBT124</td>
<td>Simulated Brightness Temperature for GOES 12, Channel 4</td>
</tr>
<tr>
<td>SBT125</td>
<td>Simulated Brightness Temperature for GOES 12, Channel 5</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SBT124</td>
<td>Simulated Brightness Temperature for GOES E Infrared</td>
</tr>
<tr>
<td>SBT123</td>
<td>Simulated Brightness Temperature for GOES E Water Vapor</td>
</tr>
<tr>
<td>SBT114</td>
<td>Simulated Brightness Temperature for GOES W Infrared</td>
</tr>
<tr>
<td>SBT113</td>
<td>Simulated Brightness Temperature for GOES W Water Vapor</td>
</tr>
<tr>
<td>SRFA161</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-1</td>
</tr>
<tr>
<td>SRFA162</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-2</td>
</tr>
<tr>
<td>SRFA163</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-3</td>
</tr>
<tr>
<td>SRFA164</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-4</td>
</tr>
<tr>
<td>SRFA165</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-5</td>
</tr>
<tr>
<td>SRFA166</td>
<td>Simulated Reflectance Factor for ABI GOES-16, Band-6</td>
</tr>
<tr>
<td>SRFA171</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-1</td>
</tr>
<tr>
<td>SRFA172</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-2</td>
</tr>
<tr>
<td>SRFA173</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-3</td>
</tr>
<tr>
<td>SRFA174</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-4</td>
</tr>
<tr>
<td>SRFA175</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-5</td>
</tr>
<tr>
<td>SRFA176</td>
<td>Simulated Reflectance Factor for ABI GOES-17, Band-6</td>
</tr>
<tr>
<td>SHTMP</td>
<td>Skin Temperature</td>
</tr>
<tr>
<td>SRCNO</td>
<td>Slight risk convective outlook</td>
</tr>
<tr>
<td>SSGSO</td>
<td>Slope Of Sub-Grid Scale Orography</td>
</tr>
<tr>
<td>SNOAG</td>
<td>Snow age</td>
</tr>
<tr>
<td>SNOAG</td>
<td>Snow Age</td>
</tr>
<tr>
<td>SALBD</td>
<td>Snow Albedo</td>
</tr>
<tr>
<td>SCE</td>
<td>Snow Cover by elevation (snow=0-252,neither=253,clouds=254)</td>
</tr>
<tr>
<td>SCP</td>
<td>Snow Cover</td>
</tr>
<tr>
<td>SC</td>
<td>Snow Cover (snow=250,clouds=100,neither=50)</td>
</tr>
<tr>
<td>SNOWC</td>
<td>Snow cover</td>
</tr>
<tr>
<td>SNOWC</td>
<td>Snow Cover</td>
</tr>
<tr>
<td>SDEN</td>
<td>Snow Density</td>
</tr>
<tr>
<td>SDEN</td>
<td>Snow Density</td>
</tr>
<tr>
<td>SNOD</td>
<td>Snow depth</td>
</tr>
<tr>
<td>SNO D</td>
<td>Snow Depth</td>
</tr>
<tr>
<td>SDWE</td>
<td>Snow Depth Water Equivalent</td>
</tr>
<tr>
<td>SDWE</td>
<td>Snow Depth Water Equivalent</td>
</tr>
<tr>
<td>SEVAP</td>
<td>Snow Evaporation</td>
</tr>
<tr>
<td>SEVAP</td>
<td>Snow Evaporation</td>
</tr>
<tr>
<td>SRWEQ</td>
<td>Snowfall Rate Water Equivalent</td>
</tr>
<tr>
<td>SRWEQ</td>
<td>Snowfall rate water equivalent</td>
</tr>
<tr>
<td>SNFALB</td>
<td>Snow-free albedo</td>
</tr>
<tr>
<td>SNFALB</td>
<td>Snow-Free Albedo</td>
</tr>
<tr>
<td>SNO M</td>
<td>Snow Melt</td>
</tr>
<tr>
<td>SNOM</td>
<td>Snow melt</td>
</tr>
<tr>
<td>SNMR</td>
<td>Snow Mixing Ratio</td>
</tr>
<tr>
<td>SNMR</td>
<td>Snow mixing ratio</td>
</tr>
<tr>
<td>SNOHf</td>
<td>Snow phase change heat flux</td>
</tr>
<tr>
<td>SNOHf</td>
<td>Snow Phase Change Heat Flux</td>
</tr>
<tr>
<td>SPRATE</td>
<td>Snow Precipitation Rate</td>
</tr>
<tr>
<td>SPRATE</td>
<td>Snow Precipitation Rate</td>
</tr>
<tr>
<td>SNOT</td>
<td>Snow temperature, depth-avg</td>
</tr>
<tr>
<td>SNOT</td>
<td>Snow temperature</td>
</tr>
</tbody>
</table>

3.1. Quick Example
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNOT</td>
<td>Snow temperature</td>
<td>K</td>
</tr>
<tr>
<td>SNOT</td>
<td>Snow Temperature</td>
<td>K</td>
</tr>
<tr>
<td>SCE</td>
<td>Snow water equivalent</td>
<td></td>
</tr>
<tr>
<td>SWEPN</td>
<td>Snow water equivalent percent of normal</td>
<td>%</td>
</tr>
<tr>
<td>SOILM</td>
<td>Soil moisture content</td>
<td></td>
</tr>
<tr>
<td>SOILM</td>
<td>Soil Moisture</td>
<td>kg/m^3</td>
</tr>
<tr>
<td>RCSOL</td>
<td>Soil moisture parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>RCSOL</td>
<td>Soil moisture parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>SOILP</td>
<td>Soil Porosity</td>
<td>Fraction</td>
</tr>
<tr>
<td>POROS</td>
<td>Soil porosity</td>
<td>Proportion</td>
</tr>
<tr>
<td>TSOIL</td>
<td>Soil temperature</td>
<td></td>
</tr>
<tr>
<td>SOILP</td>
<td>Soil Porosity</td>
<td>Proportion</td>
</tr>
<tr>
<td>EUVIRR</td>
<td>Solar EUV Irradiance</td>
<td></td>
</tr>
<tr>
<td>RCS</td>
<td>Solar parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>RCS</td>
<td>Solar parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>Solar photosphere</td>
<td></td>
</tr>
<tr>
<td>SWHR</td>
<td>Solar Radiative Heating Rate</td>
<td></td>
</tr>
<tr>
<td>SOLRF</td>
<td>Solar Radio Emissions</td>
<td></td>
</tr>
<tr>
<td>SPECIRR</td>
<td>Solar Spectral Irradiance</td>
<td></td>
</tr>
<tr>
<td>XLONG</td>
<td>Solar X-ray Flux (XRS Long)</td>
<td></td>
</tr>
<tr>
<td>XSHRT</td>
<td>Solar X-ray Flux (XRS Short)</td>
<td></td>
</tr>
<tr>
<td>SOLZA</td>
<td>Solar Zenith Angle</td>
<td></td>
</tr>
<tr>
<td>AMSL</td>
<td>Specific Altitude Above Mean Sea Level</td>
<td>m</td>
</tr>
<tr>
<td>QZ0</td>
<td>Specific humidity at top of viscous sublayer</td>
<td>kg/kg</td>
</tr>
<tr>
<td>SPF H</td>
<td>Specific Humidity</td>
<td></td>
</tr>
<tr>
<td>SPF H</td>
<td>Specific humidity</td>
<td></td>
</tr>
<tr>
<td>HTGL</td>
<td>Specified Height Level Above Ground</td>
<td></td>
</tr>
<tr>
<td>SRCS</td>
<td>Specified radius from the center of the Sun</td>
<td>m</td>
</tr>
<tr>
<td>DWWW</td>
<td>Spectral directional width of the wind waves</td>
<td></td>
</tr>
<tr>
<td>SPF FTR</td>
<td>Spectral Peakedness Factor</td>
<td></td>
</tr>
<tr>
<td>SICE D</td>
<td>Speed of ice drift</td>
<td></td>
</tr>
<tr>
<td>SPRDF</td>
<td>Spread F</td>
<td></td>
</tr>
<tr>
<td>HST DV</td>
<td>Standard deviation of height</td>
<td></td>
</tr>
<tr>
<td>SDSSGO</td>
<td>Standard Deviation Of Sub-Grid Scale Orography</td>
<td></td>
</tr>
<tr>
<td>TSD1 D</td>
<td>Standard Dev. of IR Temp. over 1x1 deg. area</td>
<td></td>
</tr>
<tr>
<td>HLCY</td>
<td>Storm relative helicity</td>
<td></td>
</tr>
<tr>
<td>SSRUN</td>
<td>Storm surface runoff</td>
<td></td>
</tr>
<tr>
<td>SURGE</td>
<td>Storm Surge</td>
<td></td>
</tr>
<tr>
<td>STPA</td>
<td>Storm total precip accum</td>
<td></td>
</tr>
<tr>
<td>STRM</td>
<td>Stream function</td>
<td></td>
</tr>
<tr>
<td>SBSNO</td>
<td>Sublimation (evaporation from snow)</td>
<td></td>
</tr>
<tr>
<td>SBSNO</td>
<td>Sublimation (evaporation from snow)</td>
<td></td>
</tr>
<tr>
<td>SUN</td>
<td>Sunshine duration (ECMWF proposal, not WMO approved)</td>
<td></td>
</tr>
<tr>
<td>SUNSD</td>
<td>Sunshine Duration</td>
<td></td>
</tr>
<tr>
<td>SUNS</td>
<td>SunShine</td>
<td></td>
</tr>
<tr>
<td>SIPD</td>
<td>Supercooled Large Droplet Icing</td>
<td></td>
</tr>
<tr>
<td>SIPD</td>
<td>Supercooled Large Droplet (SLD) Icing</td>
<td></td>
</tr>
<tr>
<td>SLDP</td>
<td>Supercooled Large Droplet (SLD) Probability see note 1</td>
<td></td>
</tr>
<tr>
<td>SLDP</td>
<td>Supercooled Large Droplet (SLD) Probability</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SuperLayerCompositeReflectivity</td>
<td>Super Layer Composite Reflectivity (33-60 kft)</td>
<td></td>
</tr>
<tr>
<td>AKHS</td>
<td>Surface exchange coefficients for T and Q divided by delta z</td>
<td></td>
</tr>
<tr>
<td>AKMS</td>
<td>Surface exchange coefficients for U and V divided by delta z</td>
<td></td>
</tr>
<tr>
<td>LFTX</td>
<td>Surface Lifted Index</td>
<td></td>
</tr>
<tr>
<td>SLI</td>
<td>Surface lifted index</td>
<td></td>
</tr>
<tr>
<td>PrecipType</td>
<td>Surface Precipitation Type (SPT)</td>
<td></td>
</tr>
<tr>
<td>SFCR</td>
<td>Surface roughness</td>
<td></td>
</tr>
<tr>
<td>SSST</td>
<td>Surface Salinity Trend</td>
<td></td>
</tr>
<tr>
<td>SLTYP</td>
<td>Surface Slope Type</td>
<td></td>
</tr>
<tr>
<td>ModelSurfaceTemperature</td>
<td>Surface Temperature</td>
<td></td>
</tr>
<tr>
<td>SSTT</td>
<td>Surface Temperature Trend</td>
<td></td>
</tr>
<tr>
<td>SSTOR</td>
<td>Surface water storage</td>
<td></td>
</tr>
<tr>
<td>Surge</td>
<td>Surge Height</td>
<td></td>
</tr>
<tr>
<td>SX</td>
<td>Sweat index</td>
<td></td>
</tr>
<tr>
<td>TMPA</td>
<td>Temperature anomaly</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Temperature</td>
<td></td>
</tr>
<tr>
<td>TMP</td>
<td>Temperature</td>
<td></td>
</tr>
<tr>
<td>TPSWP</td>
<td>Temperature</td>
<td></td>
</tr>
<tr>
<td>RCT</td>
<td>Temperature parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>RCT</td>
<td>Temperature parameter in canopy conductance</td>
<td></td>
</tr>
<tr>
<td>TTDIA</td>
<td>Temperature Tendency By All Physics</td>
<td></td>
</tr>
<tr>
<td>TTRAD</td>
<td>Temperature tendency by all radiation</td>
<td></td>
</tr>
<tr>
<td>TTRAD</td>
<td>Temperature tendency by all radiation</td>
<td></td>
</tr>
<tr>
<td>TTPHY</td>
<td>Temperature Tendency By Non-radiation Physics</td>
<td></td>
</tr>
<tr>
<td>WTEND</td>
<td>Tendency of vertical velocity</td>
<td></td>
</tr>
<tr>
<td>MASK</td>
<td>Thematic Mask</td>
<td></td>
</tr>
<tr>
<td>THICK</td>
<td>Thickness</td>
<td></td>
</tr>
<tr>
<td>TSC</td>
<td>Thunderstorm coverage</td>
<td></td>
</tr>
<tr>
<td>TSMT</td>
<td>Thunderstorm maximum tops</td>
<td></td>
</tr>
<tr>
<td>TSTM</td>
<td>Thunderstorm probability</td>
<td></td>
</tr>
<tr>
<td>TSTM</td>
<td>Thunderstorm Probability</td>
<td></td>
</tr>
<tr>
<td>TACONCP</td>
<td>Time-integrated air concentration of caesium pollutant</td>
<td></td>
</tr>
<tr>
<td>TACONIP</td>
<td>Time-integrated air concentration of iodine pollutant</td>
<td></td>
</tr>
<tr>
<td>TACONRDP</td>
<td>Time-integrated air concentration of radioactive pollutant</td>
<td></td>
</tr>
<tr>
<td>PTOR</td>
<td>Tornado probability</td>
<td></td>
</tr>
<tr>
<td>TORPROB</td>
<td>Tornado probability</td>
<td></td>
</tr>
<tr>
<td>TRNPROB</td>
<td>Tornado probability</td>
<td></td>
</tr>
<tr>
<td>TCDC</td>
<td>Total cloud cover</td>
<td></td>
</tr>
<tr>
<td>TCOLI</td>
<td>Total column-integrated cloud ice</td>
<td></td>
</tr>
<tr>
<td>TCOLI</td>
<td>Total Column-Integrated Cloud Ice</td>
<td></td>
</tr>
<tr>
<td>TCOLW</td>
<td>Total column-integrated cloud water</td>
<td></td>
</tr>
<tr>
<td>TCOLW</td>
<td>Total Column-Integrated Cloud Water</td>
<td></td>
</tr>
<tr>
<td>TCOLC</td>
<td>Total column-integrated condensate</td>
<td></td>
</tr>
<tr>
<td>TCOLC</td>
<td>Total Column-Integrated Condensate</td>
<td></td>
</tr>
<tr>
<td>TCOLOM</td>
<td>Total column-integrated melting ice</td>
<td></td>
</tr>
<tr>
<td>TCOLOM</td>
<td>Total column-integrated melting ice</td>
<td></td>
</tr>
<tr>
<td>TCOLOM</td>
<td>Total column-integrated melting ice</td>
<td></td>
</tr>
<tr>
<td>TCIOZ</td>
<td>Total Column Integrated Ozone</td>
<td></td>
</tr>
<tr>
<td>TCOLR</td>
<td>Total Column Integrated Rain</td>
<td></td>
</tr>
<tr>
<td>TCOLR</td>
<td>Total column integrated rain</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>TCOLR</td>
<td>Total Column Integrated Rain</td>
<td></td>
</tr>
<tr>
<td>TCOLS</td>
<td>Total Column Integrated Snow</td>
<td></td>
</tr>
<tr>
<td>TCOLS</td>
<td>Total column integrated snow</td>
<td></td>
</tr>
<tr>
<td>TCOLS</td>
<td>Total Column Integrated Snow</td>
<td></td>
</tr>
<tr>
<td>TCLSW</td>
<td>Total column-integrated supercooled liquid water</td>
<td></td>
</tr>
<tr>
<td>TCIWV</td>
<td>Total Column Integrated Water Vapour</td>
<td></td>
</tr>
<tr>
<td>TCIWV</td>
<td>Total Column Integrated Water Vapour</td>
<td></td>
</tr>
<tr>
<td>TCLG</td>
<td>Total Column Integrate Graupel</td>
<td></td>
</tr>
<tr>
<td>TCWAT</td>
<td>Total Column Water (Vertically integrated total water (vapour+cloud water/ice)</td>
<td></td>
</tr>
<tr>
<td>TCOND</td>
<td>Total Condensate</td>
<td></td>
</tr>
<tr>
<td>TCOND</td>
<td>Total Condensate</td>
<td></td>
</tr>
<tr>
<td>THFLX</td>
<td>Total Downward Heat Flux at Surface</td>
<td></td>
</tr>
<tr>
<td>TIPD</td>
<td>Total Icing Potential Diagnostic</td>
<td></td>
</tr>
<tr>
<td>TIPD</td>
<td>Total Icing Potential Diagnostic</td>
<td></td>
</tr>
<tr>
<td>TOZNE</td>
<td>Total ozone</td>
<td></td>
</tr>
<tr>
<td>A PCP</td>
<td>Total Precipitation</td>
<td></td>
</tr>
<tr>
<td>APCP</td>
<td>Total precipitation</td>
<td></td>
</tr>
<tr>
<td>APCPN</td>
<td>Total precipitation (nearest grid point)</td>
<td></td>
</tr>
<tr>
<td>TPRATE</td>
<td>Total Precipitation Rate</td>
<td></td>
</tr>
<tr>
<td>TPRATE</td>
<td>Total Precipitation Rate</td>
<td></td>
</tr>
<tr>
<td>PRSIGSV</td>
<td>Total Probability of Extreme Severe Thunderstorms (Days 2,3)</td>
<td></td>
</tr>
<tr>
<td>PRSIGS5VR</td>
<td>Total Probability of Extreme Severe Thunderstorms (Days 2,3)</td>
<td></td>
</tr>
<tr>
<td>PRSVR</td>
<td>Total Probability of Severe Thunderstorms (Days 2,3)</td>
<td></td>
</tr>
<tr>
<td>ASNOW</td>
<td>Total Snowfall</td>
<td></td>
</tr>
<tr>
<td>TOTSN</td>
<td>Total snowfall</td>
<td></td>
</tr>
<tr>
<td>TSRATE</td>
<td>Total Snowfall Rate</td>
<td></td>
</tr>
<tr>
<td>TSREN</td>
<td>Total Snowfall Rate</td>
<td></td>
</tr>
<tr>
<td>TSRWE</td>
<td>Total Snowfall Rate Water Equivalent</td>
<td></td>
</tr>
<tr>
<td>TSRWE</td>
<td>Total Snowfall Rate Water Equivalent</td>
<td></td>
</tr>
<tr>
<td>TSNOW</td>
<td>Total Snow</td>
<td></td>
</tr>
<tr>
<td>TSNOW</td>
<td>Total Snow</td>
<td></td>
</tr>
<tr>
<td>TSNOWP</td>
<td>Total Snow Precipitation</td>
<td></td>
</tr>
<tr>
<td>TSNOWP</td>
<td>Total Snow Precipitation</td>
<td></td>
</tr>
<tr>
<td>TTX</td>
<td>Total totals index</td>
<td></td>
</tr>
<tr>
<td>TWATP</td>
<td>Total Water Precipitation</td>
<td></td>
</tr>
<tr>
<td>TWATP</td>
<td>Total Water Precipitation</td>
<td></td>
</tr>
<tr>
<td>TTHDP</td>
<td>Transient thermocline depth</td>
<td></td>
</tr>
<tr>
<td>TRANSO</td>
<td>Transpiration Stree-Onset(Soil Moisture)</td>
<td></td>
</tr>
<tr>
<td>SMREF</td>
<td>Transpiration stress-onset (soil moisture)</td>
<td></td>
</tr>
<tr>
<td>TRANS</td>
<td>Transpiration</td>
<td></td>
</tr>
<tr>
<td>TCHP</td>
<td>Tropical Cyclone Heat Potential</td>
<td></td>
</tr>
<tr>
<td>TRO</td>
<td>Tropopause</td>
<td></td>
</tr>
<tr>
<td>TRBBS</td>
<td>Turbulence base</td>
<td></td>
</tr>
<tr>
<td>TURBB</td>
<td>Turbulence Base</td>
<td></td>
</tr>
<tr>
<td>TURB</td>
<td>Turbulence</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>TPFI</td>
<td>Turbulence Potential Forecast Index</td>
<td></td>
</tr>
<tr>
<td>TURB</td>
<td>Turbulence</td>
<td></td>
</tr>
<tr>
<td>TRBTP</td>
<td>Turbulence Top</td>
<td></td>
</tr>
<tr>
<td>TURBT</td>
<td>Turbulence Top</td>
<td></td>
</tr>
<tr>
<td>TKE</td>
<td>Turbulent Kinetic Energy</td>
<td></td>
</tr>
<tr>
<td>TKE</td>
<td>Turbulent kinetic energy</td>
<td></td>
</tr>
<tr>
<td>UOGRD</td>
<td>u-component of current</td>
<td></td>
</tr>
<tr>
<td>MAXUW</td>
<td>U Component of Hourly Maximum 10m Wind Speed</td>
<td></td>
</tr>
<tr>
<td>UICE</td>
<td>u-component of ice drift</td>
<td></td>
</tr>
<tr>
<td>UGUST</td>
<td>u-component of wind gust</td>
<td></td>
</tr>
<tr>
<td>UGRD</td>
<td>u-component of wind</td>
<td></td>
</tr>
<tr>
<td>USTM</td>
<td>U-component storm motion</td>
<td></td>
</tr>
<tr>
<td>USST</td>
<td>U-Component Storm Motion</td>
<td></td>
</tr>
<tr>
<td>USSD</td>
<td>U-component Surface Stokes Drift</td>
<td></td>
</tr>
<tr>
<td>UVI</td>
<td>Ultra Violet Index</td>
<td></td>
</tr>
<tr>
<td>UPHL</td>
<td>Updraft Helicity in Layer 2-5 km AGL</td>
<td></td>
</tr>
<tr>
<td>UPHL</td>
<td>Updraft Helicity</td>
<td></td>
</tr>
<tr>
<td>ULSM</td>
<td>Upper layer soil moisture</td>
<td></td>
</tr>
<tr>
<td>ULST</td>
<td>Upper layer soil temperature</td>
<td></td>
</tr>
<tr>
<td>ULWRF</td>
<td>Upward Long-Wave Rad. Flux</td>
<td></td>
</tr>
<tr>
<td>ULWRF</td>
<td>Upward long-wave radiation flux Upward Long-W/m^2</td>
<td></td>
</tr>
<tr>
<td>USWRF</td>
<td>Upward Short-Wave Rad. Flux</td>
<td></td>
</tr>
<tr>
<td>USWRF</td>
<td>Upward short-wave radiation flux Upward Short-W/m^2</td>
<td></td>
</tr>
<tr>
<td>UTRF</td>
<td>Upward Total radiation Flux</td>
<td></td>
</tr>
<tr>
<td>DUVB</td>
<td>UV-B downward solar flux</td>
<td></td>
</tr>
<tr>
<td>UVI</td>
<td>UV Index</td>
<td></td>
</tr>
<tr>
<td>UVIUCS</td>
<td>UV Index (Under Clear Sky)</td>
<td></td>
</tr>
<tr>
<td>VAPP</td>
<td>Vapor pressure</td>
<td></td>
</tr>
<tr>
<td>VAPP</td>
<td>Vapor Pressure</td>
<td></td>
</tr>
<tr>
<td>VOGRD</td>
<td>v-component of current</td>
<td></td>
</tr>
<tr>
<td>VOGRD</td>
<td>v-component of current</td>
<td></td>
</tr>
<tr>
<td>MAXVV</td>
<td>V Component of Hourly Maximum 10m Wind Speed</td>
<td></td>
</tr>
<tr>
<td>VICE</td>
<td>v-component of ice drift</td>
<td></td>
</tr>
<tr>
<td>VICE</td>
<td>v-component of ice drift</td>
<td></td>
</tr>
<tr>
<td>VGRD</td>
<td>v-component of wind</td>
<td></td>
</tr>
<tr>
<td>VSTM</td>
<td>V-component storm motion</td>
<td></td>
</tr>
<tr>
<td>VSTM</td>
<td>V-Component Storm Motion</td>
<td></td>
</tr>
<tr>
<td>VSSD</td>
<td>V-component Surface Stokes Drift</td>
<td></td>
</tr>
<tr>
<td>VEGT</td>
<td>Vegetation canopy temperature</td>
<td></td>
</tr>
<tr>
<td>VGTYP</td>
<td>Vegetation Type</td>
<td></td>
</tr>
<tr>
<td>VEG</td>
<td>Vegetation</td>
<td></td>
</tr>
<tr>
<td>SPEED</td>
<td>Velocity Magnitude (Speed)</td>
<td></td>
</tr>
<tr>
<td>LMV</td>
<td>Velocity Point Model Surface</td>
<td></td>
</tr>
<tr>
<td>VPOT</td>
<td>Velocity potential</td>
<td></td>
</tr>
<tr>
<td>VRATE</td>
<td>Ventilation Rate</td>
<td></td>
</tr>
<tr>
<td>VDFHR</td>
<td>Vertical Diffusion Heating rate</td>
<td></td>
</tr>
<tr>
<td>VDFVA</td>
<td>Vertical Diffusion Meridional Acceleration</td>
<td></td>
</tr>
<tr>
<td>VDFMR</td>
<td>Vertical Diffusion Moistening Rate</td>
<td></td>
</tr>
</tbody>
</table>

3.1. Quick Example 177
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDFUA</td>
<td>Vertical Diffusion Zonal Acceleration</td>
</tr>
<tr>
<td>VEDH</td>
<td>Vertical Eddy Diffusivity Heat exchange</td>
</tr>
<tr>
<td>VTEC</td>
<td>Vertical Electron Content</td>
</tr>
<tr>
<td>VII</td>
<td>Vertically Integrated Ice (VII)</td>
</tr>
<tr>
<td>VILIQ</td>
<td>Vertically-integrated liquid</td>
</tr>
<tr>
<td>MRMSVILDensity</td>
<td>Vertically Integrated Liquid (VIL) Density</td>
</tr>
<tr>
<td>MRMSVIL</td>
<td>Vertically Integrated Liquid (VIL)</td>
</tr>
<tr>
<td>VIL</td>
<td>Vertically Integrated Liquid (VIL)</td>
</tr>
<tr>
<td>VWSH</td>
<td>Vertical speed shear</td>
</tr>
<tr>
<td>VUCSHE</td>
<td>Vertical u-component shear</td>
</tr>
<tr>
<td>VVCSSH</td>
<td>Vertical v-component shear</td>
</tr>
<tr>
<td>DZDT</td>
<td>Vertical velocity geometric</td>
</tr>
<tr>
<td>VVEL</td>
<td>Vertical velocity pressure</td>
</tr>
<tr>
<td>VPTMP</td>
<td>Virtual potential temperature</td>
</tr>
<tr>
<td>VTMP</td>
<td>Virtual temperature</td>
</tr>
<tr>
<td>VIS</td>
<td>Visibility</td>
</tr>
<tr>
<td>VBDSF</td>
<td>Visible Beam Downward Solar Flux</td>
</tr>
<tr>
<td>SBSALB</td>
<td>Visible, Black Sky Albedo</td>
</tr>
<tr>
<td>VDDSF</td>
<td>Visible Diffuse Downward Solar Flux</td>
</tr>
<tr>
<td>Visible</td>
<td>Visible Imagery</td>
</tr>
<tr>
<td>SW VIL</td>
<td>Visible, White Sky Albedo</td>
</tr>
<tr>
<td>VASH</td>
<td>Volcanic ash</td>
</tr>
<tr>
<td>VAF TD</td>
<td>Volcanic Ash Forecast Transport and Dispersion</td>
</tr>
<tr>
<td>VOLASH</td>
<td>Volcanic Ash</td>
</tr>
<tr>
<td>VOL DEC</td>
<td>Volumetric Direct Evaporation Cease(Soil Moisture)</td>
</tr>
<tr>
<td>VSOSM</td>
<td>Volumetric Saturation Of Soil Moisture</td>
</tr>
<tr>
<td>SOIL W</td>
<td>Volumetric soil moisture content</td>
</tr>
<tr>
<td>VO SOIL</td>
<td>Volumetric Soil Moisture</td>
</tr>
<tr>
<td>VOLTSO</td>
<td>Volumetric Transpiration Stree-Onset(Soil Moisture)</td>
</tr>
<tr>
<td>VW LTM</td>
<td>Volumetric Wilting Moisture</td>
</tr>
<tr>
<td>WC INC</td>
<td>Water condensate added by precip assimilation</td>
</tr>
<tr>
<td>WC CONV</td>
<td>Water Condensate Flux Convergance (Vertical Int)</td>
</tr>
<tr>
<td>WC V FLX</td>
<td>Water Condensate Meridional Flux (Vertical Int)</td>
</tr>
<tr>
<td>WC U FLX</td>
<td>Water Condensate Zonal Flux (Vertical Int)</td>
</tr>
<tr>
<td>WE ASD</td>
<td>Water Equivalent of Accumulated Snow Depth</td>
</tr>
<tr>
<td>WE ASD</td>
<td>Water equivalent of accumulated snow depth</td>
</tr>
<tr>
<td>WATR</td>
<td>Water runoff</td>
</tr>
<tr>
<td>TEMP WTR</td>
<td>Water temperature</td>
</tr>
<tr>
<td>WV INC</td>
<td>Water vapor added by precip assimilation</td>
</tr>
<tr>
<td>WV CONV</td>
<td>Water Vapor Flux Convergence (Vertical Int)</td>
</tr>
<tr>
<td>WV V INC</td>
<td>Water Vapor Imagery</td>
</tr>
<tr>
<td>WV V FLX</td>
<td>Water Vapor Meridional Flux (Vertical Int)</td>
</tr>
<tr>
<td>WV U FLX</td>
<td>Water Vapor Zonal Flux (Vertical Int)</td>
</tr>
<tr>
<td>WD IRW</td>
<td>Wave Directional Width</td>
</tr>
<tr>
<td>WES P</td>
<td>Wave Energy Spectrum</td>
</tr>
<tr>
<td>WV SP 1</td>
<td>Wave spectra (1)</td>
</tr>
<tr>
<td>WV SP 2</td>
<td>Wave spectra (2)</td>
</tr>
<tr>
<td>WV SP 3</td>
<td>Wave spectra (3)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>WSTP</td>
<td>Wave Steepness</td>
</tr>
<tr>
<td>WSTR</td>
<td>Wave Stress</td>
</tr>
<tr>
<td>wxType</td>
<td>Weather</td>
</tr>
<tr>
<td>ModelWetbulbTemperature</td>
<td>Wet Bulb Temperature</td>
</tr>
<tr>
<td>WHTCOR</td>
<td>White Light Coronagraph Radiance</td>
</tr>
<tr>
<td>WHTRAD</td>
<td>White Light Radiance</td>
</tr>
<tr>
<td>WILT</td>
<td>Wilting Point</td>
</tr>
<tr>
<td>WILT</td>
<td>Wilting point</td>
</tr>
<tr>
<td>WCI</td>
<td>Wind chill factor</td>
</tr>
<tr>
<td>WDIR</td>
<td>Wind direction (from which blowing)</td>
</tr>
<tr>
<td>WMIXE</td>
<td>Wind mixing energy</td>
</tr>
<tr>
<td>WINDPROB</td>
<td>Wind probability</td>
</tr>
<tr>
<td>WGS</td>
<td>Wind speed gust</td>
</tr>
<tr>
<td>PWS</td>
<td>Wind speed</td>
</tr>
<tr>
<td>WIND</td>
<td>Wind speed</td>
</tr>
<tr>
<td>HGT X</td>
<td>X-gradient of Height</td>
</tr>
<tr>
<td>LPS X</td>
<td>X-gradient of Log Pressure</td>
</tr>
<tr>
<td>XRAYRAD</td>
<td>X-Ray Radiance</td>
</tr>
<tr>
<td>HGT Y</td>
<td>Y-gradient of Height</td>
</tr>
<tr>
<td>LPS Y</td>
<td>Y-gradient of Log Pressure</td>
</tr>
<tr>
<td>UGWDD</td>
<td>Zonal flux of gravity wave stress</td>
</tr>
<tr>
<td>U-GWD</td>
<td>Zonal Flux of Gravity Wave Stress</td>
</tr>
</tbody>
</table>

### 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

The AWIPS software package released by the Unidata Program Center is considered to be in the public domain since it is released without proprietary code. As such, export controls do not apply. Any person is free to download, modify, distribute, or share Unidata AWIPS in any form. Entities who modify or re-distribute Unidata AWIPS software are encouraged to conduct their own FOSS/COTS entitlement/license review to ensure that they remain compatible with the associated terms (see FOSS_COTS_License.pdf at https://github.com/Unidata/awips2).

### 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-

### 3.1. Quick Example

179
Fig. 1: image
environment (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.
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 terminal-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

3.1. Quick Example
awips.dataaccess.CombinedTimeQuery, 15
awips.dataaccess.DataAccessLayer, 8
awips.dataaccess.ModelSounding, 13
awips.dataaccess.PyData, 12
awips.dataaccess.PyGeometryData, 13
awips.dataaccess.PyGridData, 12
awips.dataaccess.ThriftClientRouter, 14
awips.DateTimeConverter, 15
awips.gfe.IFPClient, 15
awips.RadarCommon, 14
awips.ThriftClient, 14
awips.TimeUtil, 14
Symbols

__weakref__ (awips.dataaccess>IDataRequest attribute), 11

A
addIdentifier() (awips.dataaccess.IDataRequest method), 11
awips.dataaccess CombinedTimeQuery (module), 15
awips.dataaccess.DataAccessLayer (module), 8
awips.dataaccess.ModelSounding (module), 13
awips.dataaccess.PyData (module), 12
awips.dataaccess.PyGeometryData (module), 13
awips.dataaccess.PyGridData (module), 12
awips.dataaccess.ThriftClientRouter (module), 14
awips.DateTimeConverter (module), 15
awips.gfe.IFPClient (module), 15
awips.RadarCommon (module), 14
awips.ThriftClient (module), 14
awips.TimeUtil (module), 14

c
changeEDEXHost() (in module awips.dataaccess.DataAccessLayer), 8
changeEDEXHost() (in module awips.dataaccess.ModelSounding), 13
commitGrid() (awips.gfe.IFPClient.IFPClient method), 15
constructTimeRange() (in module awips.dateTimeConverter), 15
convertToDateType() (in module awips.dateTimeConverter), 15

D
determineDrtOffset() (in module awips.TimeUtil), 14

e
encode_dep_vals() (in module awips.RadarCommon), 14
encode_radial() (in module awips.RadarCommon), 14
encode_thresh_vals() (in module awips.RadarCommon), 14

G
gedata_type() (in module awips.RadarCommon), 14
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
gedatettime_str() (in module awips.RadarCommon), 15
getAvailableLocationNames() (in module awips.dataaccess.DataAccessLayer), 9
getAvailableParameters() (in module awips.dataaccess.DataAccessLayer), 9
getAvailableParameters() (in module awips.dataaccess.DataAccessLayer), 9
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getAvailableTimes() (in module awips.dataaccess.ThriftClientRouter), 14
getDatatype() (awips.dataaccess.IDataRequest method), 11
getEnvelope() (awips.dataaccess.IDataRequest method), 11
<table>
<thead>
<tr>
<th>Function</th>
<th>Module/Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getForecastRun()</td>
<td>aw-ips.dataaccess.DataAccessLayer</td>
<td>9</td>
</tr>
<tr>
<td>getGeometry()</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>13</td>
</tr>
<tr>
<td>getGeometryData()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getGridData()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getGridInventory()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>9</td>
</tr>
<tr>
<td>getIdentifiers()</td>
<td>awips.dataaccess.IDataRequest</td>
<td>11</td>
</tr>
<tr>
<td>getIdentifierValues()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getLatLonCoords()</td>
<td>awips.dataaccess.PyGridData</td>
<td>12</td>
</tr>
<tr>
<td>getLevel()</td>
<td>awips.dataaccess.PyData</td>
<td>12</td>
</tr>
<tr>
<td>getLevels()</td>
<td>awips.dataaccess.IDataRequest</td>
<td>11</td>
</tr>
<tr>
<td>getLocationName()</td>
<td>awips.dataaccess.PyData</td>
<td>12</td>
</tr>
<tr>
<td>getLocationNames()</td>
<td>awips.dataaccess.IDataRequest</td>
<td>11</td>
</tr>
<tr>
<td>getMetarObs()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>9</td>
</tr>
<tr>
<td>getNotificationFilter()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getOptionalIdentifiers()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getOptionalIdentifiers()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getParameter()</td>
<td>awips.dataaccess.PyGridData</td>
<td>12</td>
</tr>
<tr>
<td>getParameters()</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>12</td>
</tr>
<tr>
<td>getParmList()</td>
<td>awips.gfe.IFPClient</td>
<td>15</td>
</tr>
<tr>
<td>getRadarProductIDs()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>10</td>
</tr>
<tr>
<td>getRadarProductNames()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>10</td>
</tr>
<tr>
<td>getRawData()</td>
<td>awips.dataaccess.PyGridData</td>
<td>12</td>
</tr>
<tr>
<td>getRequiredIdentifiers()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getRequiredIdentifiers()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>10</td>
</tr>
<tr>
<td>getSelectTR()</td>
<td>awips.gfe.IFPClient</td>
<td>15</td>
</tr>
<tr>
<td>getSiteID()</td>
<td>awips.gfe.IFPClient</td>
<td>15</td>
</tr>
<tr>
<td>getSynopticObs()</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>10</td>
</tr>
<tr>
<td>getType()</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>13</td>
</tr>
<tr>
<td>getUnitTest()</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>13</td>
</tr>
<tr>
<td>getSupportedDatatypes()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>getSupportedDatatypes()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>10</td>
</tr>
<tr>
<td>IDataRequest</td>
<td>awips.dataaccess</td>
<td>11</td>
</tr>
<tr>
<td>IFPClient</td>
<td>awips.gfe.IFPClient</td>
<td>15</td>
</tr>
<tr>
<td>LazyGridLatLon</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>makeTime()</td>
<td>awips.TimeUtil</td>
<td>14</td>
</tr>
<tr>
<td>newDataRequest()</td>
<td>awips.dataaccess.ThriftClientRouter</td>
<td>14</td>
</tr>
<tr>
<td>newDataRequest()</td>
<td>awips.dataaccess.DataAccessLayer</td>
<td>10</td>
</tr>
<tr>
<td>PyData</td>
<td>awips.dataaccess.PyData</td>
<td>12</td>
</tr>
<tr>
<td>PyGeometryData</td>
<td>awips.dataaccess.PyGeometryData</td>
<td>13</td>
</tr>
<tr>
<td>PyGridData</td>
<td>awips.dataaccess.PyGridData</td>
<td>12</td>
</tr>
<tr>
<td>sendRequest()</td>
<td>awips.ThriftClient</td>
<td>14</td>
</tr>
</tbody>
</table>
setDatatype() (awips.dataaccess.IDataRequest method), 11
setEnvelope() (awips.dataaccess.IDataRequest method), 11
setLazyLoadGridLatLon() (awips.dataaccess.ThriftClientRouter.ThriftClientRouter method), 14
setLazyLoadGridLatLon() (in module awips.dataaccess.DataAccessLayer), 10
setLevels() (awips.dataaccess.IDataRequest method), 11
setLocationNames() (awips.dataaccess.IDataRequest method), 11
setParameters() (awips.dataaccess.IDataRequest method), 11

ThriftClient (class in awips.ThriftClient), 14
ThriftClientRouter (class in awips.dataaccess.ThriftClientRouter), 14
ThriftRequestException, 14