**Inventory and Monitoring Program**

**National Park Service**
**Natural Resource Stewardship and Science**

# IMD Climate Package
# User Guide and Documentation

V1.2

November 2016

# 1 Overview

The purpose of the IMD Climate Toolset is to provide all NPS staff:

- Access to weather and climate data from a consistent and NPS-sanctioned data repository(ies)
- Consistent logic for pulling and aggregating climate data, regardless of the tool being used
- Functionality tailored to the specific needs of the NPS (e.g., searching by park unit)

Development of this Toolset will be iterative. With each release, we plan to build on functionality based on defined priorities. Furthermore, we will address bugs and enhancements as the user community reports them.

Functionality is purposefully limited to facilitate iterative capabilities . We do not currently support a graphical user interface (GUI). Therefore, the current user community will be limited to more advanced developers familiar with the R or Python integrated development environments (IDE). For data sources, we focus only on the existing ACIS web services (http://www.rcc-acis.org/docs_webservices.html) for station information and station data. For functionality, the tool supports basic data request modules using consistent (identical between R and Python) logic:

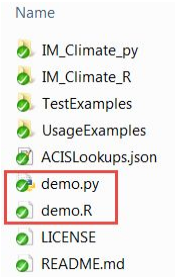
- **StationFinder** - Locate/find an inventory of stations based on one or more criteria
- **StationDataRequestor** - Request daily and/or monthly data for one or more stations and/or climate parameters
- **GridRequestor** - Requests daily gridded data (currently only PRISM) for one or more climate parameters

In all cases, users will have the option to export and save the information, including the station inventories and climate data, to their local computer. Because the logic of the R and Python[1] modules is the same, file formats are also identical between the two[2].

The following sections detail the supported functions/methods for each of these modules. For each function/method[3], we describe separately the following items:

- **Inputs** - the required and optional arguments to call the function
- **Response** - The information (in memory), returned by the function[4].
- **Outputs** - The format of the information that can be saved/exported.

Further examples of how to use the R and Python code are found in the respective demo files:




[1] To be consistent with NPS IT, functionality is tested only for Python 2.7 in a Windows environment.
[2] There are actually two very subtle differences when writing to a text file. First, R truncates whole numbers, thus 42.0 is saved as 42. Second, Python does not treat NA the same way as R and therefore puts the NA in quotes (e.g., “NA”).
[3] From this point forward we will refer to class methods as “functions” for readability
[4] While the information content will be similar between Python and R, the data structures and means of accessing this information will be tailored to each language

# 2 StationFinder

This module provides functionality that support the ability to locate an inventory of stations.

## 2.1 Function: findStation

This function helps locate all stations meeting a specified set of criteria.

**Arguments**

- **unitCode** (optional) – 4-letter unit code. Currently accepts only one.
- **distance** (optional) – Distance in kilometers for buffering a bounding box of park. If no distance is specified then 0 is used as the default buffer.
- **climateParameters** (optional) – accepts one or more of the weather parameter codes in Table 1.
- **sdate** (optional) -Start date (yyyy-mm-dd or yyyymmdd).
- **edate** (optional)- End date (yyyy-mm-dd or yyyymmdd)
- **filePathAndName** (optional)**–** If provided, a csv text file is saved to the working directory.

Table 1. ACIS weather parameter code arguments.

| Weather Parameter Code | Description |
|---|---|
| maxt | Maximum temperature (°F) |
| mint | Minimum temperature (°F) |
| avgt | Average temperature (°F) |
| obst | Obs time temperature (°F) |
| pcpn | Precipitation (inches) |
| snow | Snowfall (inches) |
| snwd | Snow depth (inches) |

### R Example

```
# Find stations collecting minimum and maximum temperature within 30km of Marsh-Billings Rockefeller NHP:

findStation(unitCode = "MABI", distance=30, climateParameters=list('mint,maxt'))
```

### Python Example

```
>>>sf = StationFinder()

#Locate all stations around MABI within a 30km buffer. Save results as MABI_Stations.csv

>>>wxStations = sf.findStation(unitCode = 'MABI', distance = 30, climateParameters = 'mint, maxt', filePathAndName = 'MABI_Stations.csv')
```

**Response**

The following station properties are available in both R (as a data frame) or Python (dictionary-like object):

- Station Properties
    - uid – The unique station identifier used by ACIS.
    - name – The station name in ACIS.
    - longitude– The longitude and latitude of the station in decimal degrees.
    - latitude – The latitude of the station in decimal degrees.
    - sid1 – Alias station ID from another observation network
    - sid1_type - Observation network for sid1
    - sid2 - Alias station ID from another observation network
    - sid2_type - Observation network for sid2
    - sid3 - Alias station ID from another observation network
    - sid3_type - Observation network for sid1
    - state – The state code for the station.
    - elev – The elevation (ft) of the station in feet.
    - minDate - The earliest observation date of the specified set of climate parameters.
    - maxDate - The most recent observation date of the specified set of climate parameters.
- Query parameters
    - unitCode – The four letter park unit code (if requested).

### R Example

```
    uid                name longitude latitude    sid1 sid1_type        sid2 sid2_type        sid3 sid3_type state elev    minDate    maxDate unitCode
1  25031     N SPRINGFIELD LAKE -72.50555 43.33916 435982 2      COOP USC00435982 6      COOP       NOSV1 7     NWSLI    VT  560 2000-01-03 2016-09-06     MABI
2  25032 SPRINGFIELD HARTNESS AP -72.51778 43.34361  54740 1      WBAN         VSF 3       FAA        KVSF 5      ICAO    VT  578 1996-07-01 2016-09-05     MABI
3  25033             CAVENDISH -72.59889 43.38472 431243 2      COOP USC00431243 6      COOP       CAVV1 7     NWSLI    VT  842 1903-02-18 2014-02-18     MABI
…
```

### Python Example

```
>>>print (wxStations)
['uid', 'name', 'longitude', 'latitude', 'sid1', 'sid1_type', 'sid2', 'sid2_type', 'sid3', 'sid3_type', 'state', 'elev', 'minDate', 'maxDate', 'unitCode']
[17605, 'WEST LEBANON', -72.31667, 43.63333, '279318 2', 'COOP', 'USC00279318 6', 'COOP', 'NA', 'NA', 'NH', 380.0, '1931-01-01', '1960-09-30', 'MABI']
[25033, 'CAVENDISH', -72.59889, 43.38472, '431243 2', 'COOP', 'USC00431243 6', 'COOP', 'CAVV1 7', 'NWSLI', 'VT', 842.0, '1903-02-18', '2014-02-18', 'MABI']
[17611, 'HANOVER', -72.28555, 43.70528, '273850 2', 'COOP', 'USC00273850 6', 'COOP', 'HANN3 7', 'NWSLI', 'NH', 584.0, '1893-06-01', '2016-09-16', 'MABI']
[25047, 'WOODSTOCK', -72.50723, 43.63028, '439984 2', 'COOP', 'USC00439984 6', 'COOP', 'WDSV1 7', 'NWSLI', 'VT', 600.0, '1892-10-01', '2016-09-19', 'MABI']
[25051, 'WILDER', -72.30806, 43.67028, '439764 2', 'COOP', 'USC00439764 6', 'COOP', 'WDRV1 7', 'NWSLI', 'VT', 365.0, '1937-04-01', '1960-09-30', 'MABI']
[25052, 'WHITE RIVER JUNCTION', -72.31667, 43.65, '94787 1', 'WBAN', '439691 2', 'COOP', 'USC00439691 6', 'COOP', 'VT', 361.0, '1930-07-01', '1948-05-31', 'MABI']
```

## Export

Exports data to a comma delimited text file (CSV)

- uid

- name
- longitude
- latitude
- sid1
- sid1_type
- sid2
- sid2_type
- sid3
- sid3_type
- state
- elev
- minDate
- maxDate
- unitCode

### R Example

```
#  Find stations collecting minimum and maximum  temperature within 30km of Marsh-Billings Rockefeller NHP:

findStation(unitCode = "MABI", distance=30, climateParameters=list('mint,maxt'), filePathAndName='MABI_Stations2.csv')
```

### Python Example

```
#export information to another file

>>>wxStations.export(filePathAndName = 'MABI_Stations2.csv')
```

# 3 StationDataRequestor

This module provides functionality for requesting the station data.

## 3.1 getDailyWxObservations

Fetches daily (unaggregated) weather station data using specified criteria.

**Arguments**:

- climateParameters - One or more climate parameters, comma delimited (See Table 1.)
- climateStations - One or more station identifiers (uids) passed either as a list, comma-delimited string, or the response object from the station finder
- sdate (optional) - Start date (yyyy-mm-dd or yyyymmdd) (default is period of record - “por”)
- edate (optional) - End date (yyyy-mm-dd or yyyymmdd) (default is period of record - “por”)
- filePathAndName (optional) - Name and location of a file to save.

### R Example

```
# Get mint and maxt temperature observations for two stations for a specified date range:

getDailyWxObservations(list('mint', 'maxt'), list(66176, 31746), "20120101", "20120201")
```

### Python Example

```
>>>dr =StationDataRequestor()
#Get wxData for two stations
>>>wxData = dr.getDailyWxObservations(climateStations = [66176, 31746], climateParameters = 'mint, maxt', startDate= '2012-01-01' , endDate = '2012-02-01'
                ,filePathAndName = 'DR01.csv')
#get data for stations returned from search in  section 2.1
>>>wxData = dr.getDailyWxObservations(climateStations = wxStations, climateParameters = 'mint, maxt', startDate= '2012-01-01' , endDate = '2012-02-01'
                ,filePathAndName = 'DR02.csv')
```

**Response**

Returns requested station data as a dictionary(-like) object in Python or data frame in R.

- uid - (see above)
- name - (see above)
- longitude - (see above)
- latitude - (see above)
- sid1 - (see above)
- sid1_type - (see above)
- sid2 - (see above)
- sid2_type - (see above)
- sid3 - (see above)
- sid3_type - (see above)
- elev - (see above)
- date: The date of variable observation (YYYY-MM-DD).

- parameter 1 (paramcode_value)  e.g. pcpn_in, maxt_F
- parameter 1 ACIS Flag 1 (paramcode_acis_flag)
- parameter 1 Source Flag (paramcode_source_flag)
- parameter n Flag 1 (ditto)
- parameter n (ditto)
- parameter n Source Flag (ditto)

### R Example

| uid | name | longitude | latitude | sid1 | sid1_type | sid2 | sid2_type | sid3 | sid3_type | state | elev | date | mint_F | mint_acis_flag | mint_source_flag | maxt_F | maxt_acis_flag | maxt_source_flag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 66176 | HARBISON MEADOW COLORADO | -105.8328 | 40.27080 | USR0000CSLP 6 | RAWS | <NA> | NA | <NA> | NA | CO | 8600.1 | 2012-01-01 | -14 | <NA> | U | 37 | <NA> | U |
| 66176 | HARBISON MEADOW COLORADO | -105.8328 | 40.27080 | USR0000CSLP 6 | RAWS | <NA> | NA | <NA> | NA | CO | 8600.1 | 2012-01-02 | -6 | <NA> | U | 42 | <NA> | U |
| 66176 | HARBISON MEADOW COLORADO | -105.8328 | 40.27080 | USR0000CSLP 6 | RAWS | <NA> | NA | <NA> | NA | CO | 8600.1 | 2012-01-03 | -5 | <NA> | U | 44 | <NA> | U |

...

### Python Example

```
#### (continued)

#view all data

>>>print (wxData)

['uid', 'name', 'longitude', 'latitude', 'sid1', 'sid1_type', 'sid2', 'sid2_type', 'sid3', 'sid3_type', 'state', 'elev', 'date', 'mint_F', 'mint_acis_flag', 'mint_source_flag', 'maxt_F', 'maxt_acis_flag', 'maxt_source_flag']

[25056, 'UNION VILLAGE DAM', -72.25778, 43.79167, '438556 2', 'COOP', 'USC00438556 6', 'COOP', 'UVDV1 7', 'NWSLI', 'VT', 460.0, u'2012-01-01', '28', 'NA', '7', '34', 'NA', '7']

[25056, 'UNION VILLAGE DAM', -72.25778, 43.79167, '438556 2', 'COOP', 'USC00438556 6', 'COOP', 'UVDV1 7', 'NWSLI', 'VT', 460.0, u'2012-01-02', '29', 'NA', '7', '43', 'NA', '7']

[25056, 'UNION VILLAGE DAM', -72.25778, 43.79167, '438556 2', 'COOP', 'USC00438556 6', 'COOP', 'UVDV1 7', 'NWSLI', 'VT', 460.0, u'2012-01-03', '16', 'NA', '7', '39', 'NA', '7']
```

## Export (to CSV)

- uid
- name
- longitude
- latitude
- sid1
- sid1_type
- sid2
- sid2_type
- sid3
- sid3_type
- state
- elev
- date
- parameter 1
- parameter 1
- parameter 1
- parameter n
- parameter n
- parameter n
- ...

### R Example

```
# Get mint and maxt temperature observations for two stations for a specified date range and export to a CSV file:

getDailyWxObservations(list('mint', 'maxt'), list(66176, 31746), "20120101", "20120201", filePathAndName='DR01.csv')
```

### Python Example

```
#export weather data

wxData.export(filePathAndName = 'DR02.csv')
```

## 3.2 getMonthlyWxObservations

Returns the monthly summaries/aggregates of weather observations for one or more stations. The following reductions (i.e., aggregations) are supported:

| Reduce Code | Aggregation |
|---|---|
| min | minimum |
| max | maximum |
| sum | total |
| mean | average |

**Arguments**

- **climateStations** - One or more station identifiers (uids) passed either as a list, comma-delimited string, or the response object from the station finder
- **climateParameters** (optional) – accepts one or more of the weather parameter codes in Table 1. If missing, uses add parameters from Table 1.
- **reduceCodes** (optional) - The method(s) used to summarize the daily observations into monthly values. Current options include min, max, sum, and mean. If none are provided, then all are returned as default.
- **sdate** (optional) -Start date (yyyy-mm-dd or yyyymmdd). If missing, default to "por".
- **edate** (optional)- End date (yyyy-mm-dd or yyyymmdd). If missing, default to "por".
- **maxMissing** (optional) - Maximum number of missing days within a month before the aggregate is not calculated and a missing value is returned (The recommended default is 1, or approximately 3.3% missing days within a month)
- **filePathAndName** (optional)**–** If provided, a csv text file is saved to the working directory.

### R Example

```
# Example
```

### Python Example

```
##get monthly summary for minimum and maximum temperature for four  stations (21635,  21636, 67081, 11652)

# from January 2014 to December 2015. Use default of maximum missing days of 1.

wxData = dr.getMonthlyWxSummaryByYear(climateStations =  [21635,  21636, 67081, 11652],

   climateParameters = 'mint, maxt', reduceCodes = 'min', sdate = '2014-01',
```

```
edate = '2015-12')
```

## Response

Returns requested monthly aggregated station data as a dictionary(-like) object in Python or data frame in R. Note: R does not support the date format YYYY-MM therefore, for monthly observations, the R data frame `date` vector is in character format.

### R Example

```
# Example

getMonthlyWxObservations(climateParameters=list('pcpn', 'avgt', 'obst', 'mint', 'maxt'), climateStations=25056, sdate="20150801", edate="20160831")
```

```
                uid          name longitude latitude    sid1 sid1_type        sid2 sid2_type    sid3 sid3_type state elev    date        pcpn_in_min        pcpn_in_min_countMissing        pcpn_in_max
pcpn_in_max_countMissing pcpn_in_sum pcpn_in_sum_countMissing pcpn_in_mean    ...

1  25056 UNION VILLAGE DAM -72.25778 43.79167 438556 2     COOP USC00438556 6     COOP UVDV1 7     NWSLI    VT  460 2015-08     0                 0     1.72              0     3.44              0
0.111          ...

2  25056 UNION VILLAGE DAM -72.25778 43.79167 438556 2     COOP USC00438556 6     COOP UVDV1 7     NWSLI    VT  460 2015-09     0                 0     1.55              0     4.34              0
0.145          ...

3  25056 UNION VILLAGE DAM -72.25778 43.79167 438556 2     COOP USC00438556 6     COOP UVDV1 7     NWSLI    VT  460 2015-10     0                 0     1.47              0     3.79              0
0.122          ...
```

### Python Example

```
#print the response object

print (wxData)
```

## Export (to CSV)

- uid
- name
- longitude
- latitude
- sid1
- sid1_type
- sid2
- sid2_type
- sid3
- sid3_type
- state
- elev
- date
- parameter 1_reduce code 1 value
- parameter 1_reduce code 1 maxMissing
- parameter 1_reduce code 2 value
- parameter 1_reduce code 2 maxMissing
- parameter 2_reduce code 1 value
- parameter 2_reduce code 1 maxMissing
- parameter 2_reduce code 2 value
- parameter 2_reduce code 2 maxMissing
- parameter n_reduce code n value
- parameter n_reduce code n maxMissing

- ...

### R Example

```
# Example:

getMonthlyWxObservations(climateParameters=list('pcpn', 'avgt', 'obst', 'mint', 'maxt'), climateStations=25056, sdate="20150801", edate="20160831", filePathAndName = "D:\\temp\\trash\\monthyObs_station25056.csv")
```

### Python Example

```
#save the data locally

wxData.export(filePathAndName = localFolder + 'monthlyData.csv')
```

## 4.0 Grid Requestor

Module that requests gridded data. Currently, only PRISM daily grids are supported (1981-present). The following parameters are supported for grids:

Table 2. ACIS weather parameter codes for climate grids.

| Weather Parameter Code | Description |
|---|---|
| maxt | Maximum temperature (°F) |
| mint | Minimum temperature (°F) |
| avgt | Average temperature (°F) |
| pcpn | Precipitation (inches) |

## getDailyGrids

Requests and downloads daily climate grids.

**Arguments**

- **sdate** - Start date (yyyy-mm-dd or yyyymmdd).
- **edate** - End date (yyyy-mm-dd or yyyymmdd).
- **unitCode** (optional) – 4-letter unit code. Currently accepts only one.
- **distance** (optional) – Distance in kilometers for buffering a bounding box of park. If no distance is specified then 0 is used as the default buffer.
- **climateParameters** (optional) – accepts one or more of the weather parameter codes in Table 2.
- **filePath** (optional)**–** If provided, an ASCII grid for each day in the specified date range are saved to the specified directory. Otherwise, ASCII grids will be saved to the working directory. Grid names follow the pattern of Source_parameter_aggregation_YYYYMMDD (e.g., PRISM_mint_dly_20150101)

### Python Example

```
gr = GridRequestor()

grids=  gr.getDailyGrids(sdate = '2015-01-01', edate = '2015-01-10', unitCode = 'GRKO',  distance = 10, climateParameters = 'mint')
```

### R Example

```
getDailyGrids(unitCode = list("AGFO"), distance=10, sdate = "20150801", edate = "20150803", climateParameters = list("mint", "maxt"))
```

## Response

### Python Example

In python, getDailyGrids returns a dictionary that is indexed by climate parameters and date. Each grid is a numpy array.

```
>>>print (grids)

{'mint': {'2015-01-10': Grid([[ 8.8,  5.8,  5.1,  4.4,  3.6,  2.6,  4.9,  5.2,  4.5],

     [ 8.8,  5.2,  4.6,  4. ,  3.4,  2.4,  3.9,  4.7,  5.2],

     [ 9.7,  7.5,  4.2,  3.5,  2.9,  2.3,  3.3,  3.8,  4.4],

     [ 8.6,  7.5,  4.6,  3.5,  2.9,  2. ,  2.1,  2.2,  3.3],

     [ 7.8,  6.9,  4.9,  3.6,  2.5,  1.8,  0.9,  1.2,  3.2],

     [ 6.9,  6.1,  4.8,  3.7,  2.1,  1.4,  1. ,  2.8,  2.6],

     [ 4.8,  4. ,  2.8,  2. ,  1.5,  1.1,  2.6,  3. ,  1.7],

     [ 1.5,  1.3,  0.2,  1. ,  1.2,  0.3,  1.3,  1.8, -0.2],

.................................................
```

### R Example

In R, getDailyGrids returns ASCII format grid/rasters(s) ordered by date.

## <u>Export</u>

Export saves each grid as an ASCII grid along with a projection file (*.prj).

### Python Example

```
>>>grids.export(filePath = 'C:\\TEMP\\')
```

**R Example**

```
getDailyGrids(unitCode = list("AGFO"), distance=10, sdate = "20150801", edate = "20150803", climateParameters = list("mint", "maxt"), filePath = "D:\\temp\\trash")
```

# 5.0 Appendices - Additional Logic

## Appendix A: General Logical

- All missing values are given NA if none exists. This applies on download as well.
- ACIS web service errors will cause the modules to terminate and report back the error. The one exception to this is when the error message is 'no data available'.

## Appendix B: StationFinder

**findStation**

- Invalid unitCode not error trapped. In other words, someone passes an an invalid code or a bounding box does not exist for a particular unit code or more than one unit code is specified, there is no logic to catch/warn the user and prevent the the request from failing
- All missing properties (e.g., station missing an elevation, sid1 does not exist, etc.) are converted to 'NA'.
- ACIS terminates requests returning thousands of stations (not sure of exact number). There is no attempt to trap for this. Because all arguments are optional, a user can potentially search for all stations in ACIS get no results.
- Climate parameters should be comma-delimited. Failure to do so may result in errors.
- The unit code supplied is used to request a bounding box (longitude and latitude) from the IRMA Unit service. This bounding box is buffered by a user-supplied distance (kilometers) which is translated into decimal degrees. The formula used to calculate the buffer in decimal degrees is : $Buffer_{DD} = Buffer_{KM} * 0.011$. Buffer distance is ignored if parkCode is not specified.
- When exporting the text file, there is no attempt to add extension and/or check for valid extensions. File delimiter is a comma.
- There is no attempt to error trap if only dDate or eDate is provided. ACIS will ignore the date request if only one is provided. Invalid date formats will also results in failure.
- ValidDateRange is caluclated using the following logic:
    - If no climate parameter specified, return max date range of all supported climate parameters (see table 1) (via background request)
    - If climate parameters are specified, return max combined date range for a station for specified set of parameters
- Station Type is assigned for sid1, sid2, sid3 only. Additional station IDs (sometimes up to 7) are ignored

## Appendix C: Station Data Requestor

**getDailyWxObservations**

- Requested climate parameters and station ids are not verified to be valid.
- Start and end dates are not checked against period of record.
- No changes are made to ACIS flag values in response (e.g. T-trace, M-missing, etc.).
- Empty responses are changed to "NA".
- The application will automatically request the following codes from Table 7 in the ACIS documentation: Flags and source flags (add = 'f,s')

- Multiple parameters should be comma-delimited. The program does remove white-space to ensure a successful call to ACIS.
- Start and end dates will need to be in format YYYY-MM-DD or YYYYMMDD. There is no attempt to correct or catch other date formats.
- If there is no daily data to export, then a file is not created
- If sdate and edate or not specified, the entire period of record is returned
- If multiple stations are requested and data does not exist for some, those stations not having weather data will be ignored on export

**getMonthlyWxObservations**

- In R, for monthly data, value vector returned as character format to accommodate missing or trace records ("M", “T”). In Python, export to CSV returns character format outputs.
- R does not support the YYYY-MM format for a Date object. Hence, for monthly observations, the date vector is in character format.

## Appendix D: Grid Requestor

**getDailyGrids**

- Since unitCode is optional yet ACIS requires a bounding box, the default bbox is '-130, 20,-50,60'
- Output grids are spatially registered to the centroid of the lower left pixel
- ACIS returns the temperature grids in fahrenheit, which differs from the source grids found here: http://www.prism.oregonstate.edu/
- When requesting grids, the default precision is to the nearest tenth (e.g., 10.1 degrees F). This precision results in about a 0.05 F difference in temperature.
- ACIS returns -999 for missing values

**Export**

- Currently the only supported format is ASCII grid format along with PRJ file