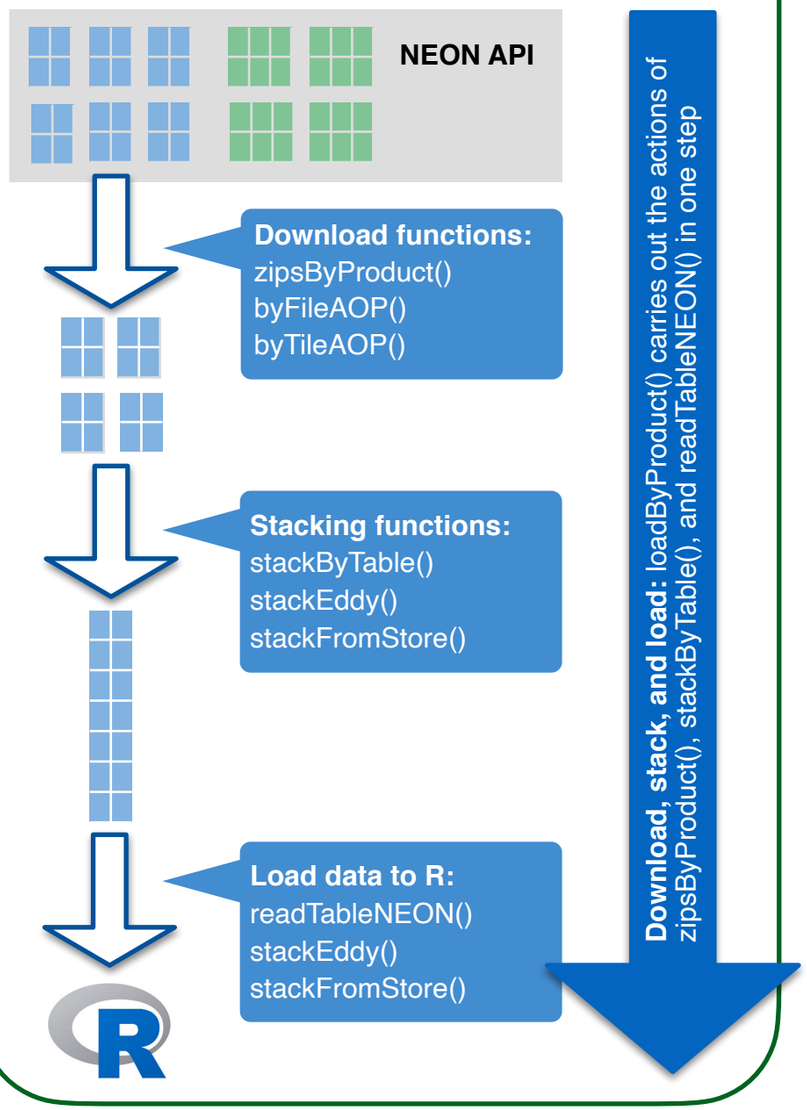
NEON data access and wrangling with neonUtilities

CHEAT SHEET Page 1

Overview

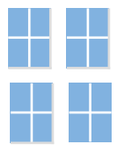
neonUtilities facilitates accessing and working with NEON data.

NEON data are published in discrete packages for each **data product**, **site**, and **month** of data collection. Each package may contain data from several **data tables** and/or **sensor locations**. neonUtilities can help you access those data packages, and also transform them to more tractable data formats.

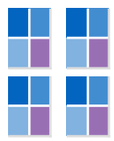


Tabular data

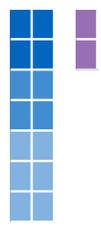
Tabular data published by NEON include both **observational** (human-collected, such as observations and measurements of birds and trees, and collection of physical samples) and **sensor** data. Data collected by sensors in the **surface-atmosphere exchange** system and the **remote sensing** platform are not tabular; see boxes below and to the right for these data.



NEON tabular data are provided in pre-packaged sets for each site and month with available data.

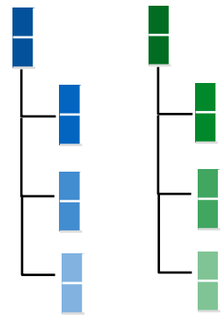


Within the site-month packages, data tables may contain the output of a single sensor, of a specific field activity (observational data), or contextual data that are relevant to all time periods (sensor position files, and observational data collected once over the lifetime of the protocol, such as trap establishment or tree mapping data).

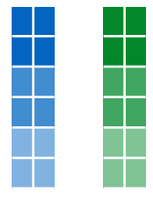


The stacking functions join the matching tables over each site and month, and over multiple sensors within a site, as necessary to create a single table for each data type. For data that are relevant to all time periods, the most recently published data are retained.

Surface-atmosphere exchange



Surface-atmosphere exchange data are published in HDF5 files: Hierarchical Data Format, in which data are arranged in a systematic hierarchy, and descriptive metadata are available at each level in the hierarchy.



At the terminal nodes in this hierarchy, the data are generally tabular. stackEddy() extracts subsets of data from the terminal nodes and joins the matching variables over months and sensors, similar to the process for the sensor data stored in simple tabular format.

Remote sensing

byFileAOP() downloads all available data for a specified **data product**, **site**, and **year**.

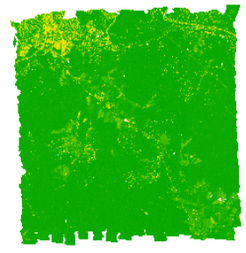
```
byFileAOP(dpID="DP3.30026.001",
  site="SCBI", year=2017)
```

SCBI: Smithsonian Conservation Biology Institute

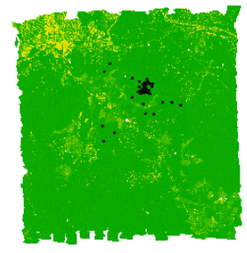
DP3.30026.001: Data product ID for Vegetation indices

For data products at the flight line level, byFileAOP() is the download function. However, for mosaicked products - available as 1 km by 1 km tiles - there is another option.

byFileAOP() downloads all available data. From the example above, here are the 143 tiles for NDVI:

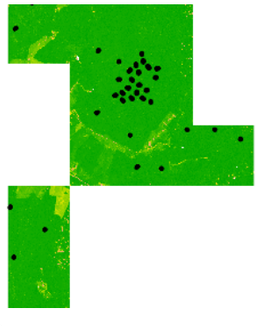


But what if you're only interested in the tiles containing locations where trees have been measured on the ground, shown here in black dots?



```
byTileAOP(dpID="DP3.30026.001",
  site="SCBI", year=2017,
  easting=veg$easting,
  northing=veg$northing)
```

Easting and northing vectors obtained from DP1.10098.001



byTileAOP() lets you download only the tiles corresponding to a specified set of coordinates, in this case only 10 out of 143 tiles!

neon Operated by Battelle

NEON data access and wrangling with neonUtilities

CHEAT SHEET Page 2

Function inputs: Download

```
zipsByProduct()
loadByProduct()
byFileAOP()
byTileAOP()
```

dpID= Data product ID of the product to download, DP#.#####.00#

site= Four-letter code for the NEON site to download. In zipsByProduct() and loadByProduct(), can be a vector of site codes, or “all"

savepath= File path to save downloaded files. Omitted in loadByProduct() since data are loaded directly to R environment; defaults to working directory if unspecified.

token= NEON API token. Optional, but if used enables faster downloads.

check.size= T/F: Require confirmation to proceed with download after calculating file size?

Underlined **input** parameters are required and do not have default values.

Only in:

```
zipsByProduct()
loadByProduct()
```

package= Basic or expanded data package

release= Data release to download. Defaults to most recent release + provisional

startdate= Start date of data to download, at 1-month resolution

enddate= End date of data to download, at 1-month resolution

timeIndex= Download a single averaging interval, by time index, for sensor data

tabl= Download a single data table, by name, for observational data

Only in:

```
byFileAOP()
byTileAOP()
```

year= Year of data to download

Only in:

```
byTileAOP()
```

easting= Vector of UTM easting coordinates used to determine which tiles to download

northing= Vector of UTM northing coordinates used to determine which tiles to download

buffer= Size of buffer around coordinates to be included when finding tiles to download

Function inputs: Data stacking

```
stackByTable()
stackEddy()
stackFromStore()
```

filepath= Path to location of files to stack. Files should be in a single directory; in stackEddy(), there is an additional option for the file path to be a vector of .h5 files.

Only in:

```
stackByTable()
```

savepath= Path to location to save stacked files. If omitted, files are saved to the filepath directory. To load to R instead of saving to file, enter “envt”.

saveUnzippedFiles= T/F: Retain the original, pre-stacked files?

nCores= Number of cores to use for parallel processing

Only in:

```
stackEddy()
```

level= Data product level (1-4) of data to extract and stack. Defaults to “dp04”, the half-hourly net fluxes.

var= Variable set to extract and stack

avg= Averaging interval to extract and stack

Only in:

```
stackFromStore()
```

pubdate= Maximum publication date of data to include in stacking

Most inputs to `loadByProduct()`, `stackByTable()`, and `stackEddy()` can be used as inputs to `stackFromStore()`: **dpID, site, package, startdate, enddate, timeIndex, level, var, nCores**

Operated by Battelle