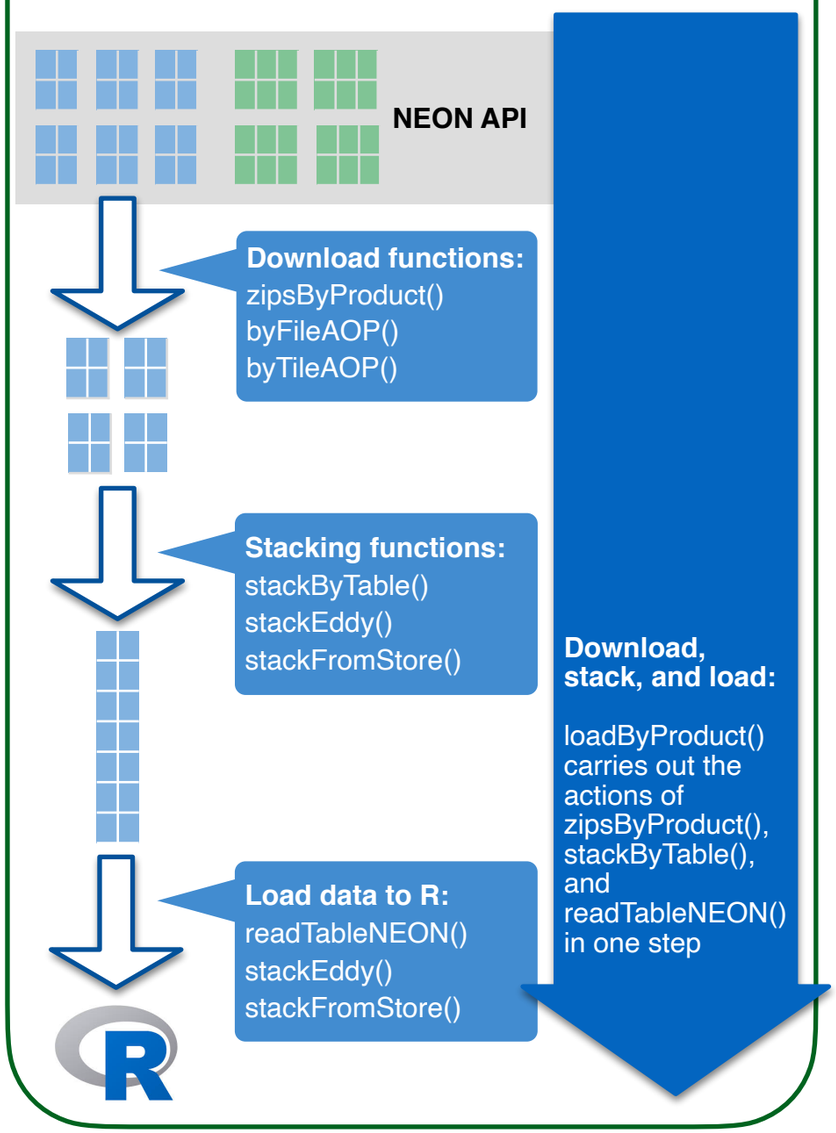
NEON data access and wrangling with neonUtilities

CHEAT SHEET Page 1

Overview

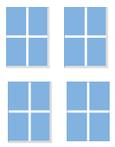
neonUtilities facilitates accessing and working with NEON data.

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.



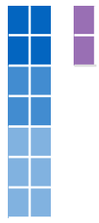
Tabular data

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.



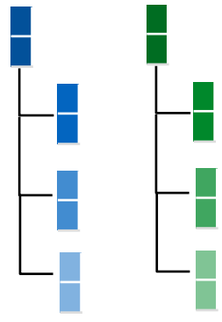
NEON tabular data are provided in pre-packaged sets for each site and month with available data.

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).

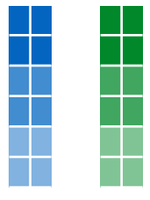


The stacking functions join the matching tables over each site and month, and over multiple sensors within a site, as necessary to create a single table for each data type. For contextual data that are relevant to all time periods, data are duplicated between site-month packages, so only the most recently published data are used in stacking.

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**.

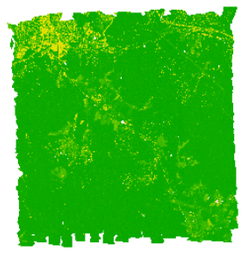
```
byFileAOP(dpID="DP3.30026.001",
  site="SCBI", year=2017)
```

SCBI: Smithsonian Conservation Biology Institute

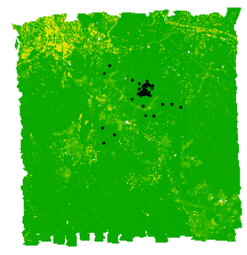
DP3.30026.001: Data product ID for Vegetation indices

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!

Operated by Battelle

NEON data access and wrangling with neonUtilities

CHEAT SHEET Page 2

Function inputs: Download

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

```
dpID="DP1.100098.001"
site=c("WREF","TOOL")
savepath="/Users/name/data"
token="alphanumeric string"
check.size=TRUE
```

- dpID: Data product ID of the product to download. IDs are listed in catalog: https://data.neonscience.org/data-products/explore
- site: Four-letter code(s) of site(s) to be downloaded, or "all" for all sites. For AOP functions, only a single site can be downloaded at once.
- 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, associated with a user account. Optional, but enables faster download.
- check.size: True/False: Require confirmation before downloading, after calculating file size?

Legend:

An example entry is given for each input parameter. Description of parameter in bubble. Underlined **input** parameters with bubbles outlined in purple are required and do not have default values.

Only in:

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

```
year=2019
```

- year: Year of data to download

Only in:

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

```
package="basic"
release="RELEASE-2021"
startdate="2019-01"
enddate="2019-12"
timeIndex=30
tabl="vst_mappingandtagging"
```

- package: Basic or expanded data package
- release: Data release to download. Defaults to most recent release + provisional
- startdate, enddate: Start and end dates of data to download, at one month resolution. Omit to download all dates.
- timeIndex: Index of time interval to download; usually equals number of minutes in averaging interval. Sensor data only.
- tabl: Table name, to download a single data table. Observational data only. Use with caution; omitting tables may leave out critical data.

Only in:

```
byTileAOP()
```

```
easting=c(583000,584000)
northing=c(5075000,5070000)
buffer=20
```

- easting, northing: Vectors of UTM easting and northing coordinates used to determine which tiles to download
- buffer: Size, in meters, of buffer around coordinates to include when determining tiles to be downloaded

Function inputs: Data stacking

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

```
filepath="/Users/name/data"
```

- filepath: Path to location of files to stack. Files should be in a single directory, except in stackEddy(), where the file path can optionally be a vector of .h5 files.

Only in:

```
stackByTable()
```

```
savepath="/Users/name/data"
saveUnzippedFiles=FALSE
nCores=1
```

- nCores: Number of cores to use for optional parallel processing
- saveUnzippedFiles: True/False: Save the original, unzipped, unstacked files?
- 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".

Only in:

```
stackEddy()
```

```
level="dp01"
var=c("rtioMoleDryCo2","dlta13CCo2")
avg=30
```

- level: Data product level of data to extract and stack. Defaults to "dp04", the half-hourly net fluxes.
- avg: Time interval to extract and stack, in minutes
- var: Variable set to extract and stack. Omit to stack all variables for the level and averaging interval.

Only in:

```
stackFromStore()
```

```
pubdate="2020-11-05"
```

- pubdate: Maximum publication date of data to include when 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