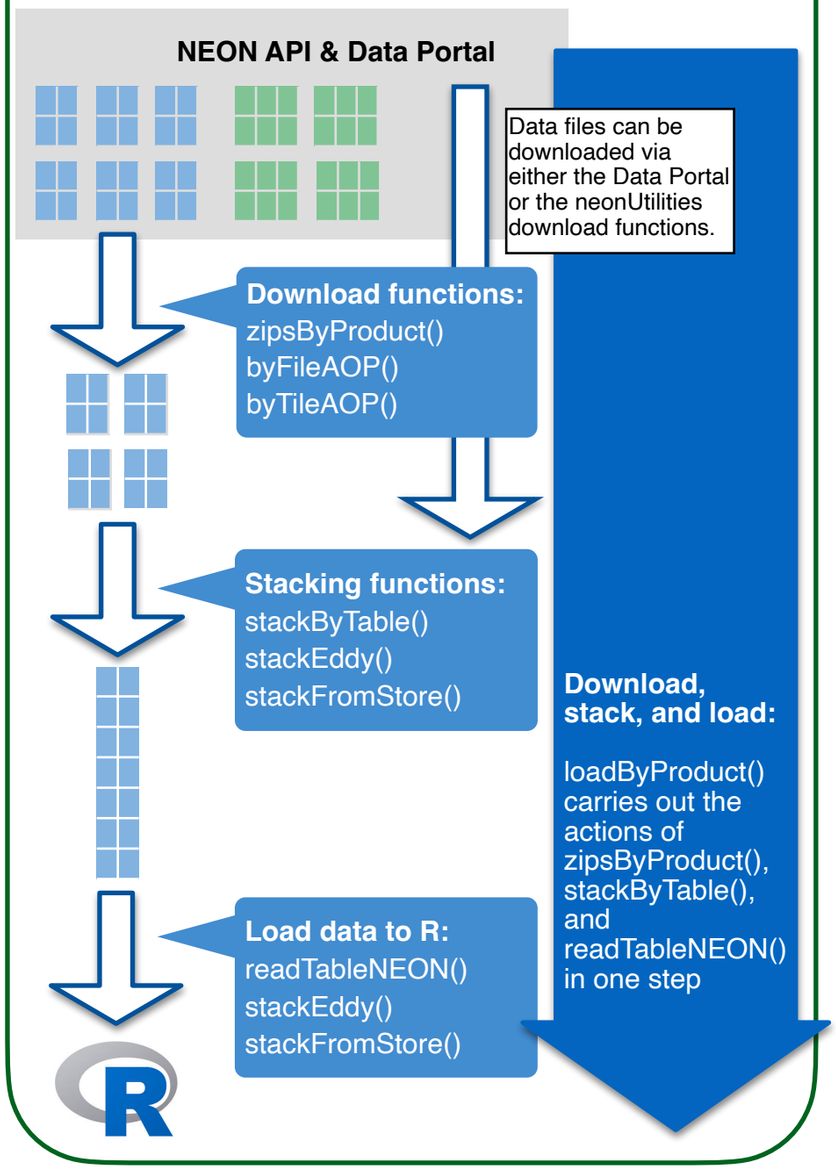
# NEON data access and wrangling with neonUtilities

CHEAT SHEET Page 1

## Overview

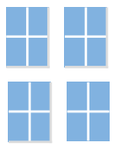
**neonUtilities** facilitates accessing and working with NEON data.

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.

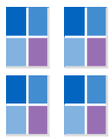



## Tabular data

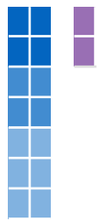
Tabular data published by NEON include both **observational** (human-collected, such as observations and measurements of birds and trees, and collection of physical samples) and **sensor** data. Data collected by sensors in the **surface-atmosphere exchange** system and the **remote sensing** platform 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.

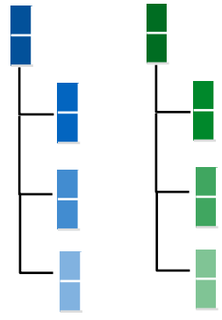


Within the site-month packages, data tables may contain the output of a single sensor, a specific field activity (observational data), or contextual data that are relevant to all time periods (sensor position files, and observational data collected once over the lifetime of the protocol, such as trap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

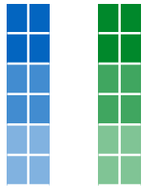


The stacking functions join the matching tables over each site and month, and over multiple sensors within a site, as necessary to create a single table for each data type. For contextual data that are relevant to all time periods, data are duplicated between site-month packages, so only the most recently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

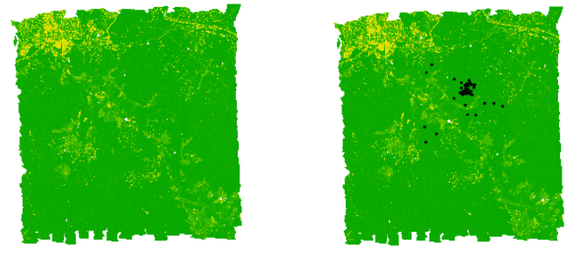
```
byFileAOP(dpID="DP3.30026.001",
  site="SCBI", year=2017)
```

**SCBI: Smithsonian Conservation Biology Institute**

**DP3.30026.001: Data product ID for Vegetation indices**

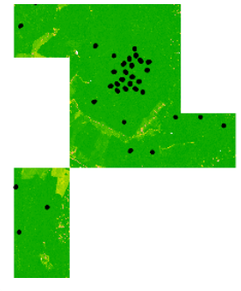
byFileAOP() downloads all available data. For example, here are the 143 tiles for the download specified above, which total 5 GB of data when uncompressed:

But downloading all 5 GB is excessive if you don't need it all. What if you're only interested in the tiles containing locations where trees have been measured on the ground, shown here in black dots? In that case, use byTileAOP() as described below.



```
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() downloads only the tiles corresponding to a specific set of coordinates, in this case only 10 out of 143 tiles!
This option is only available for mosaicked (tiled) data products.

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")
include.provisional=FALSE
savepath="/Users/name/data"
token="alphanumeric string"
check.size=TRUE
```

Data product ID of the product to download. IDs are listed in catalog: https://data.neonscience.org/data-products/explore

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.

File path to save downloaded files. Defaults to working directory if unspecified; omitted in loadByProduct() since data are loaded directly to R environment.

True/Flase: Include Provisional data in downloaded data. Provisional data are subject to change without notice; see https://www.neonscience.org/data-samples/data-management/data-revisions-releases

True/False: Require confirmation before downloading, after calculating file size?

NEON API token, associated with a user account. Optional, but enables faster download.

**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 of data to download

**Only in:**

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

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

Basic or expanded data package

Data Release to download. Defaults to most recent Release; see https://www.neonscience.org/data-samples/data-management/data-revisions-releases

Start and end dates of data to download, at one month resolution. Omit to download all dates. Download size may be very large if all dates are included.

Index of time interval to download; usually equals number of minutes in averaging interval. Sensor data only.

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
```

Vectors of UTM easting and northing coordinates used to determine which tiles to download

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"
```

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
```

Number of cores to use for optional parallel processing

True/False: Save the original, unzipped, unstacked files?

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
metadata=FALSE
useFasttime=FALSE
```

Data product level of data to extract and stack. Defaults to "dp04", the half-hourly net fluxes.

Variable set to extract and stack. Omit to stack all variables for the level and averaging interval.

Time interval to extract and stack, in minutes. Required if level is dp01.

True/False: Should the attribute data be extracted from the h5 files?

True/False: Should time stamp ingest use the fasttime package? Enables faster stacking, but may introduce uncertainty at the millisecond level.

**Only in:**

```
stackFromStore()
```

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

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

# NEON data access and wrangling with neonUtilities
CHEAT SHEET Page 3

## Best practices

### Avoid downloading the same data multiple times

loadByProduct() is the most popular choice for users downloading tabular data. But if you are writing a script you plan to run repeatedly, loadByProduct() will download the same data over and over, wasting a lot of time and compute power. Instead, use loadByProduct() to download the most recent data Release, save the data locally, and then use the local data files in your script.

For more details about data Releases and recommended workflows, see the tutorial Understanding Releases and Provisional Data.

### Manage download sizes

Many NEON data products are many gigabytes in size. Downloading and working with these data products can be challenging; they may overwhelm the compute capacity of the average computer. Suggestions for managing this challenge:

- Use byTileAOP() to download only the tiles of interest for remote sensing data
- Download remote sensing indices rather than full reflectance data sets, if the indices are sufficient for your analyses
- Download only the 30-minute data from sensor data products, if the 1-minute data aren't needed for your analyses
- For climatic data, use Summary weather statistics (DP4.00001.001)
- Consider using a cloud compute environment

### Cite the data!

When you publish a manuscript using NEON data, cite the data used in your analyses. Citations are included in the outputs of neonUtilities data-stacking functions and remote sensing download functions, in BibTeX format. Citation text is also included on the Data Product Details page for each data product (for example, https://data.neonscience.org/data-products/DP1.10026.001).

For more information about best practices for citing NEON data, documentation, and resources, see Acknowledging and Citing NEON.

## Python

The Python package neonutilities (all lowercase!) brings neonUtilities functionality to Python. There are a few differences between the R and Python packages.

### Function names and exceptions

| R | Python |
|---|---|
| `zipsByProduct()` | `zips_by_product()` |
| `stackByTable()` | `stack_by_table()` |
| `loadByProduct()` | `load_by_product()` |
| `byFileAOP()` | `by_file_aop()` |
| `byTileAOP()` | `by_tile_aop()` |
| `stackEddy()` | `Not included in initial release; planned for future development` |
| `stackFromStore()` | `Not included; neonstore integration is offered in R because neonstore is an R package` |

### Function inputs and cloud capabilities

Python stacking functions use Arrow to carry out stacking. Arrow is very fast! Thus the Python functions don't currently include a parallelization option.

Also thanks to Arrow, load_by_product() includes the input cloud_mode=. Set to True for a cloud-to-cloud transfer of data from the NEON storage buckets. Cloud mode should only be used if the destination location is in the cloud.

Just like in a local environment, running load_by_product() repeatedly to access the same data is not recommended. Save the data in the cloud environment you are working in.

We plan to implement Arrow in the R version of neonUtilities in 2025.

### Function outputs

loadByProduct() in R returns a named list of tables; load_by_product() in Python returns a dictionary of tables. The names of tables match between the two languages.